Universidad
Carlos III de Madrid

Departamento de Ingeniería de Sistemas y Automática

PROYECTO FIN DE CARRERA

I.T.I.: ELECTRÓNICA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DETECTOR DE OBJETOS PARA EL ROBOT ASIBOT

Autor: Raúl Sánchez Álvarez

Tutor: Alberto Jardón Huete

Director: Juan Carlos González Víctores

Leganés, Julio de 2011

Título: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DETECTOR DE OBJETOS PARA EL ROBOT ASIBOT

Autor: RAÚL SÁNCHEZ ÁLVAREZ

Tutor: ALBERTO JARDÓN HUETE

Director: JUAN CARLOS GONZÁLEZ VÍCTORES

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

PRESIDENTE SECRETARIO VOCAL

Agradecimientos

En primer lugar, quería dar las gracias a Alberto Jardón y a Juan González por su ayuda y apoyo en los momentos difíciles.

Junto a ellos, quiero agradecer a todos los docentes que me han formado y a todos los compañeros que me han dado su ayuda y en muchos casos su amistad.

Gracias a mis padres y mi hermano, por comprender la absorbente dedicación que le he tenido que dedicar a esta carrera y porque sin su apoyo hubiera sido muy difícil su estudio. En especial a mi padre, que siempre me animó a estudiar una carrera.

Por supuesto, gracias a todos mis amigos (en especial a Jesús y su familia) por haber estado, por estar y porque sé que estarán cuando les necesite.

Quiero dar las gracias a Marta, por su apoyo y comprensión en la última etapa de la carrera.

Gracias a todos los que me han brindado su apoyo durante todo este tiempo.

Resumen

El grupo Robotics Lab de la Universidad Carlos III de Madrid, desarrolló en 2004 un primer prototipo del robot asistencial ASIBOT, un robot cuya finalidad es la ayuda y asistencia a personas discapacitadas y de la tercera edad.

Este proyecto se ha desarrollado con el objetivo de mejorar las características funcionales del robot y facilitar su utilización al usuario. Estas mejoras surgen como línea de investigación del diseño de un sistema de visión por computador que permita detectar objetos, mostrando su ubicación dentro de una pantalla.

Para ello se ha creado una aplicación ejecutable bajo entorno Linux integrando el diseño de una interfaz gráfica para mejorar y simplificar su manejo. Dentro de esta interfaz se pueden elegir varias opciones, siendo la más importante el poder seleccionar la cámara deseada para iniciar el proceso de detección del objeto.

Abstract

The group Robotics Lab at the University Carlos III of Madrid, in 2004 developed a first prototype of the ASIBOT assistive robot, a robot whose purpose is to aid and assist disabled and elderly people.

This project has been developed with the aim of improving the functional characteristics of the robot and the user ease of use. These improvements come as a line of research in designing a computer vision system to detect objects, showing their location within a screen.

For this we have created an application that runs under Linux environment integrating the design of a graphical interface to improve and simplify its management. Within this interface several options can be choosen, the most important to select the desired camera to start the process of object detection.

Índice general

Índice de figuras

Capítulo 1

Introducción y objetivos

Según recientes datos demográficos la esperanza de vida está aumentando en los últimos tiempos de forma importante [1]. Se estima que en los países industrializados las personas mayores de 65 años tendrán en 2030 una cuota superior al 20% de la población total. Este hecho hace que parte de los desarrollos tecnológicos más avanzados estén encaminados a la mejora de la calidad de vida de estas personas. El desarrollo de sistemas robóticos dotados de un alto nivel de movilidad e inteligencia permitirá alcanzar este objetivo.

Varios robots cuyo enfoque era el de la mejora de la calidad de vida de ancianos y discapacitados fueron desarrollados en el pasado. Sin embargo, se ha visto que tienen grandes limitaciones de uso en entornos domésticos dado que necesitan de grandes y complejos sistemas de control que hacen muy difícil su transporte. Además, los robots están anclados permanentemente a una mesa o silla de ruedas limitando, de esta forma, su aplicación en diferentes habitaciones y dependencias domésticas.

Partiendo de estas premisas, se creó el primer prototipo del robot ASIBOT, realizado por el grupo Robotics Lab, en el departamento de Sistemas y Automática de la Universidad Carlos III de Madrid. El robot es capaz de adaptarse a diferentes entornos de la casa e inclusive desplazarse por la misma. El robot, por ejemplo, puede moverse por las paredes de una habitación, sobre el lavabo, estar anclado a la silla de ruedas y moverse con ella, etc. Para ello, la casa debe estar equipada con un sencillo sistema de anclajes que sirve tanto para alimentar al robot como para dotarlo de apoyo. Utilizando estos anclajes (llamados Docking Station, DS) el robot es capaz de moverse, por ejemplo, de la pared del salón a la silla de ruedas, de la silla a la encimera de la cocina, etc.

El robot ASIBOT permite realizar una gran diversidad de tareas domésticas, tales como dar de comer, afeitar, maquillar, lavar los dientes, etc. Para controlar estas tareas basta con disponer de sensores que permitan conocer la posición de cada uno de los ejes del robot. Sin embargo, actualmente se requiere que los sistemas robóticos realicen el mayor número de tareas y procesos posibles, tanto las que se controlan simplemente con una serie de sensores, como las que además necesitan la incorporación de sensores exteriores que permitan obtener parámetros relacionados con el entorno o el usuario, como por ejemplo sistemas de visión. Mediante la utilización de sistemas de visión se pueden realizar otro tipo de sistemas como por ejemplo detectores de objetos.

El campo de la detección de objetos en imágenes ha despertado en los últimos años un gran interés debido a sus posibilidades de aplicación en diversos campos, por ejemplo, para detectar rostros humanos u objetos, como es el caso de este proyecto.

La base del problema consiste en la creación de un sistema capaz de procesar una imagen y determinar si en ella se encuentra o no un determinado objeto, que constituye el objetivo de detección, y en caso afirmativo obtener las coordenadas de su localización exacta dentro de dicha imagen.

Se realizará un estudio de opciones para sistemas de detección de objetos para ver cuál es el más adecuado dentro de nuestro sistema robótico. Finalmente se diseñará una interfaz como muestra de la calidad del funcionamiento del detector, además de proporcionar a lo largo del documento datos empíricos de las pruebas realizadas.

1.1. Objetivos

El presente proyecto tiene como objetivos:

- Diseñar una aplicación informática ejecutable bajo entorno Linux que muestre la imagen de una cámara de red (cámara IP) en una ventana.

- Detectar la presencia de un objeto y señalizarlo en la ventana creada
- Incluir información de las coordenadas del objeto dentro de la pantalla.
- Desarrollar un menú que integre la visualización y detección de la presencia de objetos de varias cámaras IP, facilitando su utilización al usuario.

Para el desarrollo de este proyecto se han utilizado varias cámaras de red instaladas en el Laboratorio de Robótica Asistencial de la Universidad Carlos III de Madrid, que simula una dependencia de una casa, en concreto una cocina. Esto sirve como principio de estudio de investigación sobre los objetivos previstos. La finalidad es poder construir un sistema cámaras-detector fiable y robusto para ser utilizado con las cámaras del Laboratorio de Robótica Asistencial y además, en un futuro, sobre cámaras IP instaladas en el propio ASIBOT o en cualquier otro robot asistencial o de características similares.

A lo largo del documento se irán describiendo con todo detalle la evolución y consecución de estos objetivos, durante el tiempo que se ha trabajado en ello.

1.2. Descripción de los capítulos

En este primer capítulo se ha realizado una introducción sobre el tema a tratar en el estudio del proyecto y se han presentado los objetivos que se perseguían al inicio del mismo.

En el segundo capítulo se describirá más en profundidad el robot ASIBOT y las cámaras de red. Se hará una introducción a la visión por computador y el procesamiento de imágenes, se explicará una de las fundamentales librerías utilizadas para el desarrollo de la aplicación informática, OpenCV [2], y se detallará el proceso de entrenamiento de un clasificador para realizar el detector de objetos. Por último, se comentarán las características de los motores gráficos en general, y GTK+ [3] en particular.

En el tercer capítulo se explicará el desarrollo completo del proyecto, desde la creación de una aplicación para visualizar la imagen de una cámara de red hasta la aplicación final intentando alcanzar los objetivos del presente estudio. En la primera parte del capítulo se describirán detalladamente esos objetivos, y en la segunda todos los pasos seguidos hasta alcanzarlos.

En el capítulo cuarto se mostrarán y comentarán los resultados obtenidos tras finalizar el desarrollo del proyecto y se explicará mediante imágenes el funcionamiento de la aplicación diseñada.

Por último, en el capítulo quinto se finalizará el documento realizando un análisis crítico, presentando las conclusiones a las que se ha llegado en el desarrollo del proyecto y la propuesta de algunas alternativas interesantes a tener en cuenta para la realización de futuros desarrollos.

Capítulo 2

Estado del arte

En este capítulo se hará una descripción detallada de los elementos e instrumentos fundamentales necesarios para el desarrollo del proyecto. Primero se describirán los objetos físicos: el robot y la cámara de red; posteriormente una introducción a la visión por computador y procesamiento de imágenes y la principal librería utilizada para la visión por computador, OpenCV [2]; se detallará el proceso de entrenamiento de un clasificador para un detector de objetos mediante Haartraining [4]; y por último, el motor gráfico GTK+ [3].

2.1. Robot ASIBOT

De forma muy genérica, se podría describir ASIBOT como un robot asistencial de 5 grados de libertad de estructura simétrica que cuenta en cada uno de sus extremos con un mecanismo de anclaje a la DS con un alcance algo superior a un brazo humano (del orden de 1,3 m) y puede transportar un peso de hasta 2 kg en su extremo [5]. Todo el sistema de control (computadora, electrónica, transmisiones, etc.) está embebido en el robot y su peso no supera los 13,5 kg. Este bajo peso permite que el robot pueda ser fácilmente transportado por una sola persona o asistente a otra dependencia o inclusive a otro piso para poder ser usado por otros usuarios. De esta manera, el uso, los costes y los beneficios sociales pueden ser compartidos por varios usuarios a la vez.

El robot ASIBOT se puede clasificar como escalador, ya que está diseñado para trabajar desde diferentes posiciones, allá donde tenga una DS, y poder moverse de una a otra. También podría ser clasificado como manipulador, porque una vez anclado, sus operaciones principales son coger y mover cosas o posicionar herramientas, tareas típicas

de los robots manipuladores. En cuanto a su uso, está enfocado a tareas de servicio, más concretamente las asistenciales.

Para interactuar con ASIBOT se realiza mediante comunicaciones inalámbricas. La comunicación hombre robot puede realizarse mediante distintas formas según las diferentes capacidades del usuario: por voz, en caso de carecer de movilidad en las manos, por el manejo de un sencillo joystick en caso de discapacidades en los brazos, por un lápiz táctil en PDA, en el caso de discapacidad en las extremidades inferiores, con el dedo, etc. El dialogo hombre-robot se efectúa mediante un sencillo sistema de menús orientados a la tarea basado en iconos gráficos.

En la figura 1 podemos ver una imagen del ASIBOT anclado en una DS de una plataforma.

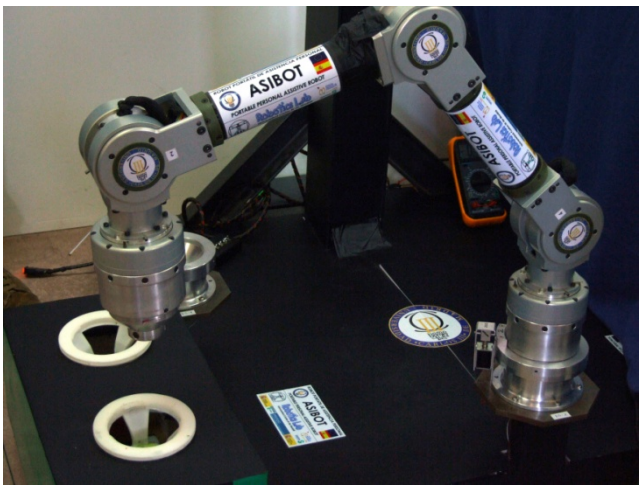

Figura 1 – Robot ASIBOT

Se han desarrollado otros proyectos y prototipos de robots con objetivos similares al ASIBOT, algunos de ellos son el robot Manus [6] y el Kares [7]. A continuación se va a realizar una pequeña descripción de las características más importantes de cada uno de ellos, ya que en un futuro se podría integrar el sistema desarrollado en este proyecto en estos sistemas robóticos.

- Manus

El robot Manus, ver figura 2, es un sistema basado en un manipulador montado sobre una silla de ruedas que ha conseguido un cierto liderazgo en Europa. Construido en Holanda por el centro de investigación TNO, el sistema consiste en un manipulador que emplea actuadores eléctricos, montado sobre una silla de ruedas adaptada para soportar dicho robot y la interfaz para controlarlo. Para manejar tanto la silla como el robot se emplean controles individuales seleccionables y procedimientos asistidos por computador; normalmente se emplea un joystick junto a un pequeño teclado de selección de opciones, ambos controlados por el movimiento residual del usuario. Los dos requisitos de diseño más importantes son la compactibilidad, para que el robot no entorpezca el movimiento del usuario, y el alcance del manipulador, que llega incluso a poder coger algo que esté en el suelo. Opera en entornos no estructurados con muy poca o nula información a priori, con lo que son de suma importancia los procedimientos de control de la pinza por parte del usuario.

Figura 2 – Robot Manus

- KARES

El robot KARES (KAIST Rehabilitation Engineering Service system) de Corea, consiste en un robot de aluminio de seis grados de libertad movido con motores paso a paso y montado sobre una silla de ruedas. Realiza las tareas de dar de comer y de beber, recoger cosas del suelo, afeitar al usuario, encender y apagar interruptores, abrir y cerrar

puertas, y tareas propias de oficina. En principio, estas tareas han sido probadas en un entorno semi-estructurado, pero para alcanzar una mayor inteligencia se tiene conocimiento del entorno a través de un sistema de visión artificial y sensores de fuerza/par ubicados en el punto extremo del robot. El usuario maneja el robot mediante un dispositivo de diez teclas montado a un lado de la silla, también se puede emplear un sistema de reconocimiento de voz.

2.2. Cámara de red (cámara IP)

Una cámara IP es una cámara que emite las imágenes directamente a la red (intranet o internet) sin necesidad de un ordenador .

Las cámaras IP han sufrido una gran evolución desde sus inicios. La primera webcam del mundo se instaló en la cafetera de la Universidad de Cambridge para monitorizar el nivel de café. Posteriormente, tomando como referencia la webcam, se desarrolló la cámara de red. Actualmente se utilizan en numerosas aplicaciones y entornos, siendo las más importantes la seguridad y vigilancia y la monitorización remota.

La característica más relevante de las cámaras IP es que permiten al usuario tener una cámara en una localización y ver el video en tiempo real desde otro lugar a través de la red o de internet.

2.2.1. Tecnología y funcionamiento de la cámara de red

Una cámara de red tiene su propia dirección IP y características propias de ordenador para gestionar la comunicación en la red [8]. Todo lo necesario desde la adquisición hasta el envío de la imagen a través de la red está incorporado en la propia cámara.

Básicamente una cámara IP se compone de: una cámara de video tradicional (lentes, sensores, procesador digital de imagen, etc.); un sistema de compresión de imagen, para reducir el número de datos y permitir una transferencia más eficiente a través de la red; y

un sistema de procesamiento encargado de la gestión de las imágenes.

La resolución de las imágenes digitales se mide en píxeles. La imagen más detallada es la que tiene más datos y por tanto mayor número de píxeles. Las imágenes con más detalles ocupan más espacio en los discos duros y precisan mayor ancho de banda para su transmisión. Para almacenar y transmitir imágenes a través de una red los datos deben estar comprimidos o consumirán mucho espacio en disco o mucho ancho de banda. Si el ancho de banda está limitado la cantidad de información que se envía debe ser reducida. Existen varios estándares de compresión, por ejemplo el JPEG o el MPEG.

2.2.2. Aplicaciones

Las cámaras de red pueden emplearse en diferentes aplicaciones, siendo las más frecuentes la seguridad y vigilancia y la monitorización remota.

- Seguridad y Vigilancia

  Las cámaras de red se usan en sistemas de seguridad, ya que permiten vídeo a tiempo real. Se pueden utilizar en sistemas industriales, como instalaciones o negocios, y en domésticas, por ejemplo en viviendas.

  Además de la vigilancia mediante la visión a tiempo real, se pueden emplear para el control de accesos a una estancia. Puede grabarse junto con la imagen de la cámara, por ejemplo, información de la fecha y la hora permitiendo en caso necesario una sencilla revisión y localización.

- Monitorización remota

  Otra aplicación, además de la vigilancia y seguridad, es la monitorización de la imagen de la cámara. Esto puede utilizarse, por ejemplo, para conseguir que una página web resulte más dinámica e interesante, y por tanto, atraer más visitas. Cada día es más frecuente promocionar zonas turísticas, lugares emblemáticos de

ciudades, etc., implementando en sus páginas web imágenes de cámaras, pudiendo incluso ver el estado meteorológico de la zona.

2.2.3. Cámara de red, Axis 207MW

La cámara de red AXIS 207 ofrece una buena calidad de imagen incluso en condiciones de poca luz [9]. También proporciona una buena eficiencia del ancho de banda gracias a su implementación de MPEG-4. El micrófono integrado permite a los usuarios remotos no solo ver, sino también escuchar en un área y aumentar las opciones de monitorización.

Las características técnicas de la cámara se detallan en la "Tabla 1", correspondiente al Anexo I del documento.

2.3. Visión por computador y procesamiento de imágenes

La finalidad de la visión artificial es la extracción de información del mundo físico a partir de imágenes, utilizando para ello un computador.

2.3.1. Aplicaciones

La visión por computador tiene numerosas aplicaciones, a continuación se describen las más importantes:

- Biomedicina: Son muy numerosas las aplicaciones médicas de la visión artificial. Se pueden destacar el análisis de imágenes tomadas por rayos x, análisis de imágenes tomadas por ultrasonidos y la aplicación en los análisis de sangre.

- Identificación: Es un campo importante dentro de la visión por computador ya que nos permite detección e identificación de caras, objetos, identificación de huellas dactilares, etc..

- Robótica: Para el guiado de robots industriales y la navegación de robots móviles.

- Agricultura: Análisis de imágenes de plantaciones tomadas por satélites para poder hacer seguimiento de los cultivos y observar posibles enfermedades en las plantas.

- Seguridad: Como ya se ha comentado, esta es una de las aplicaciones actuales más importantes de la visión por computador, entre sus funciones se puede destacar: la vigilancia de edificios, detección de explosivos, etc.

- Controles de Calidad: Es muy útil para realizar controles de calidad a diferentes productos.

2.3.2. Etapas de un sistema de visión por computador

La visión por computador intenta reproducir el procesamiento de imágenes del ser humano: captura de luz a través de los ojos, procesado e interpretación de la escena en el cerebro y actuación en consecuencia. Por tanto se pueden definir varias fases, a continuación se pasarán a describir las seguidas en este estudio.

2.3.2.1. Captura de imágenes

Esta fase consiste en la adquisición de imágenes, para ello los sensores realizan una transformación de la intensidad de luz que reflejan los objetos a cargas eléctricas, generando una señal eléctrica de video. Posteriormente se digitaliza la señal generando una

imagen digital interpretable por el ordenador. La resolución de una imagen digital se mide en pixeles, donde cada pixel representa un valor binario del tono de gris o de color de la imagen.

2.3.2.2. Reconocimiento de características y clasificación

En esta etapa se identifican las características que distinguen a cada uno de los objetos que pueden aparecer en una imagen [10]. La estructura general de un clasificador puede observarse en la figura 3. A partir de la imagen original o procesada se obtienen una serie de características (forma, color, etc.) y/o descriptores (momentos, etc.) que definen cada objeto. Por otro lado, se habrá obtenido una base de datos que contiene los modelos de los objetos que puedan aparecer en la imagen y se realizarán una serie de hipótesis sobre los posibles candidatos que puedan aparecer en la imagen.

Finalmente se debe diseñar un clasificador que reconozca el objeto a la vista de la información extraída de la imagen y los posibles candidatos.

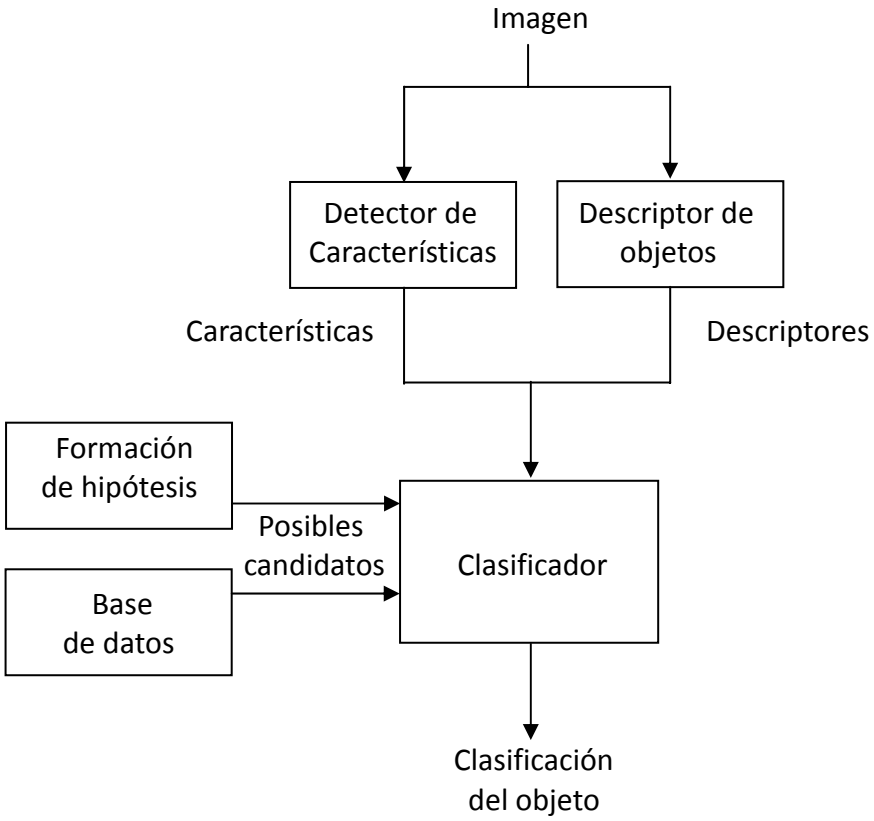

Figura 3 – Esquema general del reconocimiento de objetos

Los métodos de clasificación más empleados son los estadísticos y los sintácticos. Los estadísticos se basan en la determinación y uso de funciones de probabilidad. Los más utilizados dentro de este tipo son los paramétricos, como la Regla de Bayes [11], y los no paramétricos, como el método k-nn [12]. El teorema de Bayes vincula la probabilidad de A dado B con la probabilidad de B dado A y el método k-nn está basado en un entrenamiento mediante ejemplos cercanos al objeto en el espacio.

Los modelos sintácticos se basan en encontrar las relaciones estructurales que tienen los objetos de estudio. Se establece una analogía entre esas estructuras y la sintaxis de un lenguaje, cuyo análisis sintáctico se utiliza para clasificar los objetos.

2.3.3. Histograma de una imagen

El histograma de una imagen es una gráfica que representa los niveles de intensidad de color de dicha imagen con respecto al número de píxeles presentes con cada intensidad de color.

La ecualización del histograma es un método que se utiliza para lograr una distribución más uniforme entre el número de píxeles asociado a cada nivel de intensidad. Es decir, que exista el mismo número de píxeles para cada nivel de gris del histograma de una imagen monocroma. Es una forma de manipulación de histogramas que reduce el contraste en las áreas muy claras o muy oscuras de la imagen.

En teoría, la aplicación de esta operación debería transformar el histograma en otro con una forma perfectamente uniforme sobre todos los niveles de gris. Sin embargo, en la práctica esto no se va a poder conseguir pues se estaría trabajando con funciones de distribución discretas en lugar de continuas.

2.4. Librería OpenCV

La librería de visión artificial y código abierto The Open Computer Vision Library (OpenCV) fue creada en el año 2000 por un grupo de investigadores de Intel Corporation y desde Octubre del 2008 es patrocinada por Willow Garage [13]. Proporciona un marco de trabajo de alto nivel para el desarrollo de aplicaciones de visión por computador en tiempo real y además es gratuito [14].

Existen muchos paquetes de procesamiento de imágenes no comerciales en el mercado, por ejemplo: OpenCV, Gandalf, TargetJr e ImageLib. OpenCV tiene varias ventajas: proporciona librerías de datos estáticos y dinámicos, herramientas para trabajar con la mayoría de las cámaras del mercado, entornos de desarrollo fáciles y la posibilidad de poder ejecutarse en dos de los sistemas operativos más utilizados: Microsoft Windows y Linux.

La librería OpenCV es una API (Application Programming Interface) de aproximadamente 300 funciones escritas en lenguaje C que se caracterizan, además de las ventajas numeradas anteriormente, por lo siguiente:

- Su uso es libre tanto para su uso comercial como no comercial.

- No utiliza librerías numéricas externas, aunque puede hacer uso de alguna de ellas, si están disponibles, en tiempo de ejecución.

- Es compatible con The Intel Processing Library (IPL) y utiliza The Intel Integrated Performance Primitives (IPP) para mejorar su rendimiento, si están disponibles en el sistema. Sin embargo, la licencia por las que se rigen estas aplicaciones no es software libre.

- Dispone de interfaces para algunos otros lenguajes y entornos, por ejemplo para Matlab.

2.4.1. Estructura y características de la librería OpenCV

La librería OpenCV está dirigida fundamentalmente a la visión por computador en tiempo real. Entre sus muchas áreas de aplicación destacarían: interacción hombre-máquina; segmentación y reconocimiento de objetos; reconocimiento de gestos; seguimiento del movimiento; estructura del movimiento; y robots móviles.

El conjunto de funciones suministradas por la librería OpenCV se agrupan en los siguientes bloques:

- Estructuras y operaciones básicas: matrices, grafos, árboles, etc.
- Procesamiento y análisis de imágenes: filtros, momentos, etc.
- Análisis estructural: geometría, procesamiento del contorno, etc.
- Análisis del movimiento y seguimiento de objetos: plantillas de movimiento, seguidores, flujo óptico, etc.
- Reconocimiento de objetos: objetos propios), modelos HMM, etc.
- Calibración de la cámara: morphing, geometría epipolar, estimación de la pose, etc.
- Reconstrucción tridimensional (funcionalidad experimental): detección de objetos, seguimiento de objetos tridimensionales, etc.
- Interfaces gráficos de usuarios y adquisición de video.

2.4.2. Interfaces gráficas y herramientas de la librería OpenCV

La librería OpenCV proporciona varios paquetes de alto nivel para el desarrollo de aplicaciones de visión. Todos ellos se pueden agrupar en librerías de C/C++ dirigidas a usuarios avanzados, HighGUI [15] y CvCam; y en herramientas de scripting dirigidas a usuarios de nivel medio, Hawk y OpenCV Toolbox para Matlab.

HighGUI permite la escritura/lectura de imágenes en numerosos formatos, la captura de stream de video de cámaras/capturadoras y la creación de ventanas para visualizar imágenes en ellas, entre otras características.

CvCam nos proporciona un único interfaz de captura y reproducción bajo Linux y Win32 y callbacks para la gestión de stream de vídeo o ficheros AVI.

Hawk es un entorno visual que proporciona soporte para OpenCV, IPL y HighGUI.

Por último, la librería OpenCV proporciona un toolbox para Matlab, utilizando sus tipos nativos (matrices, estructuras).

2.5. Método de entrenamiento de un clasificador

Un sistema de detección de objetos en imágenes consiste en dividir primero la imagen en subventanas, y después evaluar cada una para averiguar si contiene o no al objeto que buscamos. La forma en que se decide si esa subventana contiene al objeto depende del método que se utilice, pero el sistema de detección siempre se basará en buscar ciertas características propias del objeto. La aplicación de dicho método se basa en unas reglas para decidir si la subventana en cuestión es el objeto o no. La clave está en que con una de estas reglas no se podría nunca construir un sistema de detección porque fallaría demasiado, pero si se combinan las que se vea que funcionan mejor, se podrá conseguir un sistema detección de muy buena calidad.

Para que un sistema de detección de objetos se considere de buena calidad, se han de tener en cuenta básicamente dos valores: la probabilidad de detección y la probabilidad de falsa alarma [16]. La primera de ellas se obtiene como el número de objetos que ha detectado el sistema frente al número de objetos que hay realmente en un conjunto de imágenes. Este valor será el que cualquier sistema de detección de objetos intentará maximizar. En cuanto a la probabilidad de falsa alarma, se obtendrá como el número de detecciones incorrectas que ha realizado el sistema, es decir, el número de veces que el sistema ha encontrado un objeto donde realmente no lo había frente al número total de detecciones. Y ese valor es el que siempre se intentará minimizar, además teniendo en cuenta que ambos valores son directamente proporcionales, con lo que si uno de ellos aumenta, el otro también lo hará.

Para construir un detector de objetos se requiere de un proceso que se denomina entrenamiento, que se puede asimilar como un aprendizaje de la máquina sobre el objeto que tendrá que detectar en las imágenes. El entrenamiento consistirá en encontrar las reglas que mejor clasifiquen al objeto y combinarlas para formar cada etapa del detector.

El Boosting [17], como se denomina un método de entrenamiento de un clasificador, se encargará de probar cada una de estas reglas sobre un conjunto de imágenes ejemplo del objeto, muestras positivas, y sobre otro conjunto de imágenes que no contengan al objeto, muestras negativas, y elegir las mejores.

Cada regla va a permitir clasificar cada muestra en función de si ésta encaja mejor o peor con ella. Una vez reunidas las mejores, es decir, las que menor error presenten se utilizarán para construir el detector de objetos de forma que se consiga una buena tasa de detección. Esto se basará en darles mayor importancia a una reglas sobre otras dependiendo de si presentaban un error menor o mayor respectivamente.

Dentro del conjunto muestral que se proporcionará, todos los ejemplos del objeto no serán iguales, sino que tendrán diferentes características de enfoque, iluminación, etc., por tanto, una misma regla no tendrá exactamente el mismo comportamiento para todos esos ejemplos. Habrá en algunos que se obtenga un error ligeramente mayor que para el resto.

Esto sirve para hacer una clasificación de todas las muestras de las que disponemos del objeto en más fáciles o más difíciles de clasificar, se puede representar matemáticamente en forma de un valor, de un peso. Se recomienda dotar de un peso mayor a aquellos ejemplos que son más difíciles de hacer encajar con las reglas simples. De este modo, se fuerza a que la máquina aprendiz centre su atención en estas imágenes ejemplo más que en las que siempre encajan bien. Con lo que el resultado final será robusto y capaz de generalizar ante diferentes variedades de visualización del objeto.

Una vez terminado el entrenamiento, se dispondrá de un clasificador de objetos prácticamente de uso inmediato.

2.5.1. Las reglas débiles

Vamos a ver en qué consisten en concreto las reglas débiles, también denominadas hipótesis débiles, en las que se fundamentará toda la detección.

Se van a evaluar áreas rectangulares de una imagen del objeto y a compararlas entre sí. Supongamos que un área es ABCD y otro área distinto CDEF. Esta regla, que se va a llamar h_t, se calcularía como indica en la ecuación 1.

$$h_t = zona_{ABCD} - zona_{CDEF} \quad (1)$$

Donde $zona_{ABCD}$ es la suma de los valores de todos los píxeles que comprende el rectángulo que encierran los puntos A, B, C y D. Y de la misma forma para $zona_{CDEF}$ y los puntos C, D, E y F.

El valor que se obtenga al realizar este cálculo se comparará con un umbral, que habrá sido elegido al probar la regla débil sobre todo el conjunto de ejemplos del objeto, conjunto muestral, y comprobar cuál era el valor óptimo que permitiría decidir si la subventana evaluada mediante esta regla era o no el objeto.

Si la evaluación de la regla supera el umbral entonces se afirmará que no se trata del

objeto, y si queda por debajo entonces se dará la detección como positiva. En muy pocos casos el resultado del cálculo será cero, esto quiere decir que la aplicación de una regla débil sobre una subventana tiene error nulo.

Se podrían construir reglas diferentes a ésta simplemente haciendo el cálculo a diferentes escalas de esta misma estructura (siempre tomando como límite las dimensiones máximas de las imágenes del objeto que constituyan el conjunto muestral) o combinando más rectángulos, o rotando todas éstas de forma vertical.

Se pueden diferenciar varias reglas en cuanto al número de áreas rectangulares de las que se pueden componer:

Dos rectángulos: Consiste en la diferencia entre la suma de los píxeles comprendidos en ambos rectángulos. Se pueden dar cuatro posibles casos si la rotamos 90, 180 o 270º.

Tres rectángulos: Se calcula sumando los píxeles englobados dentro de los dos rectángulos exteriores y restando al resultado el rectángulo interior. También podemos ponerla en vertical.

Cuatro rectángulos: Se obtiene restando la suma de cada par de rectángulos de las dos diagonales.

A éstas, que son las mismas que utilizaron Viola&Jones [18] en sus investigaciones, se puede añadir otro tipo; la que estará formada por un único rectángulo.

El cálculo de estas reglas supone un coste computacional ligeramente elevado y esto es un inconveniente para el sistema, ya que retardará la ejecución del detector sobre cada imagen. Para evaluar una imagen la dividirá en subventanas que serán sobre las que se calcularán estas reglas y además las subventanas de una misma imagen se calcularán para varias escalas, por tanto, deben poderse calcular con el menor número de operaciones posible. En este sentido tiene un papel muy importante, como se verá en la siguiente sección, una forma de codificar la imagen denominada imagen integral que reduce el

cálculo de las reglas débiles como las que hemos visto a unas pocas referencias a un array.

2.5.2. La imagen integral

La imagen integral es una forma de representar una imagen para conseguir una evaluación de las hipótesis débiles muy rápida.

La imagen integral permite evaluar las características de tipo rectangular, de una forma muy rápida a muchas escalas diferentes. Además, como se verá, su cálculo es muy sencillo ya que requiere un número muy reducido de operaciones por píxel.

Para obtenerla se debe aplicar la ecuación 2; la imagen integral en el punto (x, y) de una imagen contiene la suma de todos aquellos píxeles que queden por arriba y a la izquierda de dicho punto (incluyendo también en la suma el punto en cuestión):

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \quad (2)$$

Donde *ii(x,y)* es la imagen integral e *i(x', y')* es la imagen original. En la figura 4 se puede ver gráficamente el significado de esta expresión.

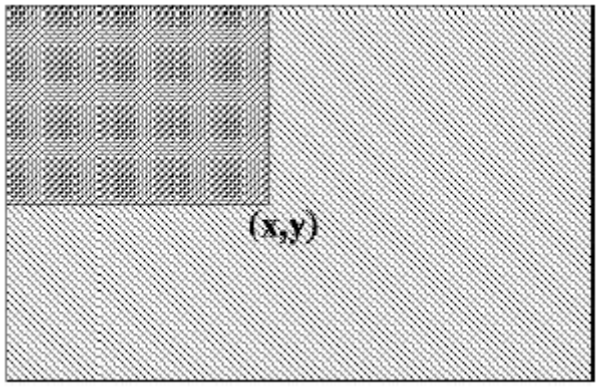

Figura 4 – Método de cálculo de la imagen integral

Para su cálculo podemos utilizar la ley recursiva indicada en la ecuación 3.

$$s(x,y) = s(x,y-1) + i(x,y)$$
$$ii(x,y) = ii(x-1,y) + s(x,y) \quad (3)$$

Donde *s(x, y)* es la suma de las intensidades de los píxeles de la fila en la que se encuentra el punto hasta y, con s*(x, −1)* = 0 e *ii(−1, y)* = 0. De este modo, la imagen integral para cualquier punto se puede calcular en un sólo paso sobre la imagen original.

Utilizando esta representación, cualquier suma rectangular dentro de la imagen se puede calcular simplemente con cuatro referencias. Y la diferencia entre dos sumas de dos rectángulos se puede obtener simplemente haciendo referencia a la imagen integral de ocho rectángulos como máximo. Además, teniendo en cuenta que en las características formadas por dos rectángulos, éstos son adyacentes, el resultado de la evaluación se puede obtener únicamente con seis referencias. Serían ocho si la característica es de tres rectángulos y nueve si está formada por cuatro de ellos.

Se puede ver esto último de manera gráfica en la figura 5, donde la suma de los píxeles del rectángulo D se puede obtener únicamente con cuatro referencias a array. El valor de la imagen integral en el punto 1 es la suma de los píxeles del rectángulo A. El valor en el punto 2 es A + B, en el punto 3 es A + C, y en el 4 es A + B + C + D. La suma dentro de D se puede obtener como 4 + 1 − (2 + 3).

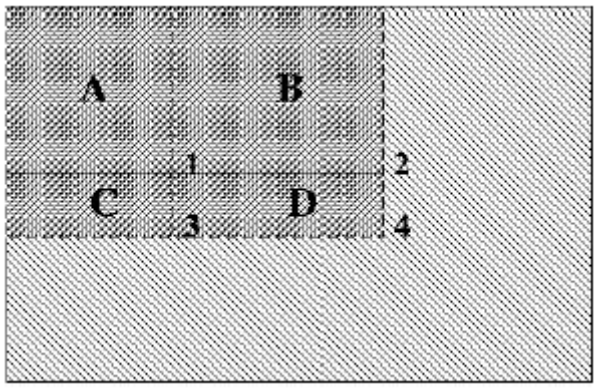

Figura 5 – Ejemplo de cálculo del valor de la imagen integral

2.5.3. Tipos de entrenamiento

Dentro del método de entrenamiento, Boosting, el tipo más utilizado para construir un clasificador de un detector de objetos es el denominado Adaboost [19]. Debe su nombre a que se ajusta adaptativamente a los errores de las reglas de clasificación débiles que genera el algoritmo evaluadas sobre la distribución de muestras o ejemplos positivos.

El Adaboost toma como entrada un conjunto de m imágenes muestra, como se indica en la ecuación 4.

$$S = [(x_1, y_1), \dots, (x_m, y_m)] \quad (4)$$

Donde x_I es una de esas imágenes ejemplo del objeto que se quiere detectar e y_I es el peso de la muestra x_I. Se asumirá para este caso que los pesos a los que puede pertenecer una muestra son:

$$Y \in [-1,1]$$

El método de entrenamiento accede al algoritmo de aprendizaje débil mediante numerosas llamadas, en cada una de las cuales se evalúa una regla débil sobre todas las imágenes del conjunto muestral y por tanto, se las clasifica en función de los pesos obtenidos para esa regla débil, así que están distribuidas de determinada forma porque unas habrán resultado más complicadas que otras. Si h_t representa una hipótesis o regla débil perteneciente a la llamada numero t del método de entrenamiento al algoritmo, entonces, el error vendrá dado por la ecuación 5.

$$\varepsilon_t = P_i[h_t(x_i) \neq y_i] \quad (5)$$

Siendo P_i la probabilidad para la muestra i de no pertenecer a la clase que afirma el clasificador o hipótesis débil. El algoritmo se quedará por tanto, en cada vuelta del método de entrenamiento, con la hipótesis que mejor clasifique la distribución de muestras, la que menor error presente. Además, la distribución muestral en la que se basa el algoritmo para

calcular el error, se va recalculando a cada vuelta, para ir dándole mayor peso a aquellas muestras mal clasificadas por la hipótesis de la vuelta anterior.

Este mismo proceso continuará durante un total de T llamadas al algoritmo. Al final, todas las hipótesis débiles obtenidas, serán combinadas en una única ley final.

Según la manera de obtener la distribución de las muestras y la forma de combinar todas las leyes para obtener una única, se pueden distinguir diferentes algoritmos de Adaboost, que se han ido proponiendo desde diferentes líneas de investigación.

Las tres variantes principales con las que se experimentan en la actualidad son:

- Discrete Adaboost (DAB)
- Real Adaboost (RAB)
- Gentle Adaboost (GAB)

2.5.3.1. Discrete Adaboost (DAB)

Esta variante inicial del Adaboost se resume en los pasos siguientes:

- Entrada:
  - Secuencia de m ejemplos de imágenes
  - Algoritmo de entrenamiento
  - T, número de iteraciones del método

1. Inicializar los pesos de las muestras que componen la distribución de m ejemplos, con $w_i = 1/m$ y repetimos para i=1,...,m.

2. Llamar al algoritmo proporcionándole la distribución de muestras

3. Evaluar la hipótesis débil, h_t

4. Calcular el error para dicha hipótesis

5. Calcular el parámetro βt, aplicando la ecuación 7

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \quad (7)$$

6. Actualizar los pesos de la distribución D_t, según la ecuación 8

$$D_{t+1}(i,y) = \frac{D_t(i,y)}{Z_t}\, \beta_t^{\left(\frac{1}{2}\right)(1+h_t(x_i,y_i)-h_t(x_i,y))} \quad (8)$$

Donde Z_t es una constante de normalización elegida de modo que la suma de los pesos siga valiendo la unidad.

- Salida:

  - La hipótesis final cuya expresión será la ecuación 9

$$h_{fin} = argmax \sum_{t=1}^{T} (log \frac{1}{\beta_t}) h_t(x,y) \quad (9)$$

La evaluación de la hipótesis débil cuando se utilice, se comparará con un umbral (*th*) escogido en función de las muestras del objeto proporcionadas para decidir si una determinada subventana es el objeto o no.

En el DAB inicialmente todos los ejemplos presentan pesos iguales. Para obtener la distribución de pesos que utilizará la vuelta del método *t + 1*, se partirá de la que había en la vuelta anterior multiplicada por un valor β_t que hará que el valor del peso de la muestra x_i aumente si h_t la ha clasificado incorrectamente, y al contrario en el caso de que la

clasificación haya sido correcta. Después los pesos son renormalizados, con lo que los ejemplos sencillos, los cuales son correctamente clasificados por las primeras reglas débiles, van quedando en un segundo plano para darles una mayor importancia (con pesos mayores) a aquellas muestras que resultan difíciles, y que por tanto suelen ser incorrectamente etiquetadas por el clasificador.

El peso de cada regla débil h_t se define como $\log(1/\beta_t)$, por tanto tendrán mayores pesos aquellas hipótesis que presentasen menor error durante la ejecución del algoritmo. Finalmente, h_{fin} permitirá etiquetar el candidato como positivo o negativo en función de si la evaluación proporciona un resultado por encima o por debajo del umbral *th*.

2.5.3.2. Real Adaboost (RAB)

Esta variante del Adaboost es similar a la anterior, simplemente cambia la forma de actualizar los pesos, que es la indicada en la ecuación 10.

$$D_{t+1}(i,y) = D_t(i,y)\exp\left(y_i * h_t(x_i,y_i)\right) \quad (10)$$

2.5.3.3. Gentle Adaboost (GAB)

La mejora de esta variante del Adaboost con respecto a los anteriores tipos se fundamenta principalmente en una carga computacional menor y en una probabilidad de detección mayor a unos valores de velocidad similares.

El GAB utiliza como entradas los mismos parámetros que las demás variedades, su diferencia está en la forma de ir actualizando los pesos, ver ecuación 11.

$$\omega_{i,t+1} = \omega_{i,t}\exp\left(-y_i h_t\right) \quad (11)$$

2.5.4. Construcción de clasificadores eficientes

Para poder evitar el alto coste computacional que supone la utilización de los detectores obtenidos, también llamados clasificadores, se puede aplicar un método para combinarlos en una estructura en cascada o árbol. Esto incrementará la velocidad del detector y permitirá su uso en sistemas de tiempo real. El detector va progresivamente centrando su atención en las zonas que parecen más prometedoras dentro de toda la imagen, y por tanto se reserva el procesado más complejo y con mayor coste computacional sólo a esas zonas seleccionadas.

Una cascada de clasificación la podemos entender como un conjunto de clasificadores sencillos que se agrupan formando diferentes etapas. Si los primeros clasificadores de ese árbol dan una respuesta positiva a la subimagen de entrada, empieza el procesado de esa subimagen con la segunda etapa, y así hasta el final del árbol. Una respuesta negativa conduce a un rechazo inmediato de la subimagen.

Para construir el árbol de decisión de esta manera, los clasificadores de las etapas finales se entrenan utilizando aquellas imágenes que los de las etapas previas han dejado pasar. Con lo que cada etapa va realizando una tarea cada vez un poco más compleja que la inmediata anterior.

2.5.5. Entrenamiento de una cascada de clasificadores

El proceso de diseño de una cascada de detección se caracteriza por unas determinadas tasas de detección y de falsa alarma que se desean conseguir. Se consideran buenas tasas de detección valores entre 85% y 90% y de falsa alarma del orden de 10^{-5}.

Dada una cascada de clasificadores, la tasa de falsa alarma de toda la cascada vendrá dada por la ecuación 12.

$$F = \Pi_{i=1}^{K} f_i \quad (12)$$

Siendo K el número de etapas y f_i la tasa de falsa alarma de cada una de las etapas por separado. Y aplicando la ecuación 13 se obtiene la tasa de detección.

$$D = \prod_{i=1}^{K} d_i \quad (13)$$

Siendo d_i la tasa de detección de cada una de las etapas de la cascada.

Una vez elegidos los valores deseados para las probabilidades de detección y falsa alarma se conocerán las condiciones que deben cumplir las etapas por separado. Por ejemplo, una tasa de detección de 0,9 se puede conseguir con 10 etapas que presenten una tasa de detección de 0,99, ya que $0{,}99^{10} \approx 0{,}9$.

La mayoría de clasificadores en cascada con un elevado número de reglas débiles, consiguen mejores resultados, sin embargo es evidente que requerirán un mayor coste computacional. En principio, los parámetros que más afectarían a las características de un clasificador serían:

- El número de etapas de la cascada
- El número de hipótesis débiles que forman cada etapa
- El umbral de cada etapa (*th*)

Para construir un clasificador eficiente se seleccionan las tasas minimas f_i y d_i. Cada etapa de la cascada se entrena mediante el Adaboost y se van añadiendo reglas débiles empleadas hasta alcanzar los valores para las probabilidades que se hayan elegido.

El proceso detallado en lenguaje natural, quedaría de la siguiente forma:

1. Se seleccionan los valores para f_i , d_i , F y D

2. Se parte de dos conjuntos; uno con muestras positivas y otro con negativas; P y N

3. Se inicializan $F_0 = 1$; $D_0 = 1$; $i = 0$

4. Repetir mientras $F_i > F$:

   - i ++

   - $n_{hipótesis\_débiles} = 0$; $F_i = F_{i-1}$

     - $n_{hipótesis\_débiles} = n_{hipótesis\_débiles} + 1$

     - Usar P y N para entrenar un clasificador con $n_{hipótesis\_débiles}$ mediante Adaboost

     - Evaluar la cascada actual obtenida mediante el conjunto de muestras de prueba para determinar F_i y D_i

     - Disminuir el umbral de la etapa i hasta que el clasificador actual tenga una tasa de detección de al menos $d_i * D_{i-1}$

   - Si $F_i > F$ entonces evaluar la cascada de clasificación actual sobre el conjunto de imágenes que no contenían al objeto y colocar algunas de las subventanas mal clasificadas, que por tanto han supuesto falsas alarmas en el conjunto N.

2.5.6. Construcción de un árbol de detección

En el apartado anterior se explica cómo en una estructura en cascada, para cada etapa, un cierto porcentaje de candidatos se rechaza y el resto son los que pasan a la siguiente etapa de detección. En esta sección se va a describir una estructura que permita construir detectores orientados a objetos con una mayor variabilidad.

Se puede interpretar como un árbol cuyas ramas estarán formadas por varios clasificadores. El entrenamiento de un árbol de estas características partirá del nodo raíz, éste se diferencia del resto de nodos en que no depende de ningún otro nodo padre.

El algoritmo es un procedimiento recursivo. En cada nodo, todas las muestras positivas y negativas transmitidas por el nodo padre, se utilizan para entrenar un clasificador. Después de entrenar el nodo padre, el árbol se empieza a dividir en estas ramas llamadas splits. Para el entrenamiento de cada una, se empleará el total de muestras negativas y la parte correspondiente de positivas que habrían resultado de dividir las que provenían del nodo padre en tantos subconjuntos como ramas se hayan desplegado. El algoritmo recursivo de construcción de las ramas se repetirá sucesivamente hasta que se alcance la profundidad requerida para los objetivos de detección especificados.

Una vez construido el árbol el proceso de clasificación de una subventana será el siguiente: el candidato empezará siendo evaluado por el nodo raíz (el de la primera etapa), si éste ya lo rechaza la subimagen será etiquetada como negativa y el proceso de detección habrá finalizado. Si por el contrario el nodo raíz acepta la muestra, ésta pasará a una de las ramas en las que se habrá dividido el nodo raíz. Si la rama da al candidato como negativo, la muestra probará suerte por la otra rama y así hasta encontrar un camino de aceptación que permita a la muestra alcanzar el final de árbol y pasar a la siguiente etapa. Si las supera todas entonces será clasificada como positiva. Si por el contrario, en algún punto del árbol ya están explorados todos los caminos posibles y la muestra ha sido rechazada por todos, ya se etiquetará como negativa y se terminará el proceso.

Con esta estructura se consiguen una tasas de detección y falsa alarma de $F \leq K f_i^K$ y $D \geq d_i^K$, donde K es el número de etapas.

Este incremento respecto a los resultados con una cascada de clasificación, se pueden compensar entrenando un número superior de etapas, para calcularlo se debe aplicar la ecuación 14.

$$\Delta K = \frac{\log\left(\frac{1}{K}\right)}{\log\left(f_i\right)} \quad (14)$$

Es lógico pensar que así se aumenta el coste computacional, sin embargo en la práctica no ocurre así ya que estas etapas extra se evaluarán en raras ocasiones, así que su contribución al coste se podrá considerar despreciable.

2.6. Programa de entrenamiento de un clasificador

La aplicación que forma parte del SDK (Software Development Kit) de OpenCV para el desarrollo del entrenamiento de un clasificador es la llamada Haartraining. Además, son necesarias las funciones que constituyen OpenCV para poder llevar a cabo con éxito tanto la ejecución de la aplicación principal como su compilación.

El programa de entrenamiento, Haartraining, implementa todo un proceso de construcción de los clasificadores de objetos que es considerablemente complejo tanto de entender como en su ejecución. Ésta presenta coste computacional muy elevado, tanto que dependiendo de los tamaños de los objetos y de las características, puede llegar a durar más de una semana en un PC doméstico.

Haartraining.cpp constituye la capa superior de la aplicación. Su código es muy sencillo ya que únicamente se encarga de recoger todos los parámetros que ha introducido el usuario en la llamada desde el interfaz de comandos y entonces llamar a la función que realmente es la principal de la aplicación, llamada cvCreateCascadeClassifier. Las acciones de esta función se van a describir ahora más detalladamente:

1. Reserva de Memoria: En primer lugar, se reserva la memoria necesaria para albergar a todo el clasificador con el número de etapas que se hayan solicitado desde la llamada. Se realizan las inicializaciones necesarias en la estructura de datos que lo contendrá, llamada CvCascadeHaarClassifier, y además se asignan las funciones encargadas de evaluar la cascada con las imágenes y de liberar la memoria reservada al final de todo el proceso. La función encargada de estas tareas es icvCreateCascadeHaarClassifier.

2. Imágenes de las muestras negativas en vector: El fichero que contiene las muestras negativas que se pasa desde la llamada, se abre ahora para lectura. Se cuentan todas las imágenes que contiene y se almacenan todas en un vector. En esta tarea intervienen las funciones icvInitBackgroundReaders e icvCreateBackgroundData.

3. Preparación de la matriz con todas las muestras de entrenamiento: Todas las muestras, tanto positivas como negativas que se emplearán para construir el clasificador, se almacenan en una gran estructura de datos compuesta por matrices y campos numéricos. El tamaño de las matrices vendrá dado por el número de muestras de las que se disponga y contendrán, desde las imágenes en sí hasta sus pesos o sus factores de normalización. Los campos numéricos son datos como las dimensiones de las muestras, o su número. Esta estructura de datos llamada CvHaarTrainingData, es la que se va a manejar durante todo el entrenamiento para extraer la información para el aprendizaje.

4. Construcción de las reglas débiles: En este momento, se construyen todas las reglas débiles posibles para el tamaño de la muestra que se ha elegido desde la llamada al programa. Esto significa que se empiezan a construir rectángulos desde un tamaño mínimo que se elige también como parámetro, hasta alcanzar el tamaño de la muestra, tanto en alto, como en ancho. Esto supone una gran cantidad de reglas débiles que el proceso tendrá que manejar y evaluar para elegir las mejores con las que construir cada etapa. La función que realiza este trabajo es icvCreateIntHaarFeatures.

   A partir de este momento, se entra en un bucle cuyas acciones construyen una de las etapas del clasificador. En cada una de las vueltas se construye (o si ya está hecha se carga en memoria) una etapa.

5. Comprobación de la existencia de la etapa: El Haartraining es capaz de seguir entrenando un mismo clasificador que ya contiene alguna etapa. Si se da este caso, el programa comenzaría su ejecución con la construcción de la etapa siguiente a la última que ya hay construida. Así en este paso el programa comprueba si la etapa

ya existe. En ese caso, pasa directamente a la siguiente. Si no existe, prepara el directorio que la albergará. La función encargada es icvLoadCARTStageHaarClassifier.

6. Obtención de las muestras positivas: Se abre el fichero que contiene las muestras positivas, se comprueba su número y su tamaño. Tras esto, se van evaluando una a una para contar solo las que obtienen un resultado diferente de cero de la cascada. La función principal que lleva a cabo este contaje es la icvGetHaarTrainingDataFromVec.

7. Búsqueda de los candidatos negativos. A partir de las imágenes que se proporcionan en el fichero de fondos desde la llamada, se han de extraer los candidatos negativos que tendrán el mismo tamaño que las muestras positivas. Para su búsqueda se hace lo siguiente: van cargando las imágenes del fichero una a una, y se parte desde el origen ($x_0 = 0$, $y_0 = 0$) de cada imagen. A partir de este punto se toma una subventana del tamaño de la muestra y se evalúa con la cascada de clasificación. Si el resultado es distinto de cero, esa subventana se computa como candidato negativo. Después se pasa a evaluar la siguiente subventana, cuyo origen se situará en (x_0 + ancho + 1, y_0) y se hará lo mismo. Así, se barren todas las x, para cada y, con pasos iguales al ancho y al alto respectivamente, de modo que al final se barre toda la imagen extrayendo los candidatos negativos que existen en ella. Al terminar de barrer una imagen se pasa a la siguiente y se hace lo mismo. Y al finalizar toda la lista de imágenes de fondo de las que se disponía se vuelve a empezar por la primera de ellas pero esta vez se toma como origen ($x_0 = 1$, $y_0 = 1$), de forma que no se vuelven a repetir los candidatos vistos en la ronda anterior. Esta búsqueda finaliza en el momento en el que se llega a reunir el número de candidatos negativos que se le han solicitado al Haartraining desde la llamada.

   Esta tarea de obtención de los candidatos negativos, se repite al inicio de la construcción de cada etapa, sin embargo, el programa tiene memoria y empieza a buscar en el punto en el que se quedó, es decir, que si hasta la etapa anterior había dado cuatro iteraciones a la lista de imágenes de fondo, y se había quedado por determinada subventana, la búsqueda comienza justo con la siguiente a esa última,

el proceso no empieza desde el principio cada vez. Así para cada etapa, consigue información nueva con la que pueda continuar el aprendizaje. Se ve aquí la importancia de proporcionar un fichero con un número de imágenes de fondo suficiente como para que el programa pueda encontrar en ellas un numero de candidatos igual al número de etapas que se van a construir por el número de candidatos negativos que se le han solicitado desde la llamada. Es decir, se ha de cumplir que, $n_{candidatostotales} \geq n_{etapas} n_{candidatossolicitados}$. Si no hay tantas imágenes, el programa empieza a barrer subventanas que ya utilizó, con lo que será más complicado extraer información, y por tanto el coste computacional que requerirá el Haartraining para finalizar el entrenamiento irá incrementándose de etapa en etapa mucho más que en el caso de que haya suficientes.

8. Asignación de los pesos: Es necesario ir otorgando más importancia a unas muestras sobre las otras para que el clasificador vaya centrando su aprendizaje en las más difíciles. Sin embargo, inicialmente, todas tienen pesos iguales. El peso que se les asigna a cada una, tanto a las positivas como a las negativas sigue la expresión definida en la ecuación 15.

$$peso = \frac{1}{nneg+npos} \quad (15)$$

siendo *nneg* y *npos* el número de muestras positivas y de negativas que se especifica en la llamada al programa. En el caso de estar construyendo la etapa 0, los pesos de las muestras se dejan con este valor único para todas, pero para cualquier otra etapa, se reasigna para conseguir el efecto explicado arriba. De ello se encarga la función llamada icvPrecalculate que utiliza las etapas que ya hay construidas para evaluar todo el conjunto muestral y ordenarlo según su dificultad para su correcta clasificación.

9. Elección y combinación de las mejores reglas débiles: En este punto del programa ya está todo preparado para la aplicación del método de entrenamiento en esencia; se dispone del espacio muestral ordenado, de la memoria necesaria para almacenar la etapa y de todo el espacio de reglas débiles posibles para el tamaño de muestra concreto. Por tanto, ahora entran en juego las funciones que aplican el método para

así seleccionar las mejores características de las disponibles y combinarlas de la forma que mejor clasifiquen al conjunto muestral concreto del que se dispone en la etapa en la que nos encontramos. La función principal que termina devolviendo las características con sus pesos que formaran la etapa es icvCreateCARTStageClassifier.

10. Guardamos la etapa recién construida: Sólo resta almacenar la etapa que se termina de construir en el archivo denominado AdaBoostCARTHaarClassifier.txt del directorio correspondiente a la etapa.

11. Finalización: Este proceso se repetirá para todas las etapas que se tengan que construir, y al finalizar la última se liberará toda la memoria reservada al inicio y el programa finalizará.

2.7. Resultados de entrenamiento

Para analizar los resultados obtenidos para los distintos entrenamientos de clasificadores realizados se van a construir las curvas ROC (acrónimo de Receiver Operating Characteristic o Característica Operativa del Receptor) [20]. En la teoría de detección de señales una curva ROC es una representación gráfica de la sensibilidad frente a (1 – especificidad) para un sistema clasificador binario según se varía el umbral de discriminación. Otra interpretación de este gráfico es la representación de la razón de verdaderos positivos (RVP = Razón de Verdaderos Positivos) frente a la razón de falsos positivos (RFP = Razón de Falsos Positivos) también según se varía el umbral de discriminación (valor a partir del cual decidimos que un caso es un positivo). ROC también puede significar Relative Operating Characteristic (Característica Operativa Relativa) porque es una comparación de dos características operativas (RVP y RFP) según se cambia el umbral para la decisión. En español es preferible mantener el acrónimo inglés, aunque es posible encontrar el equivalente español COR. No se suele utilizar ROC aislado, debemos decir “curva ROC” o “análisis ROC”.

El análisis de la curva ROC, o simplemente análisis ROC, proporciona herramientas para seleccionar los modelos posiblemente óptimos y descartar modelos subóptimos. La curva ROC se desarrolló por ingenieros eléctricos para medir la eficacia en la detección de objetos enemigos en campos de batalla mediante pantallas de radar, a partir de lo cual se desarrolla la Teoría de Detección de Señales (TDS). El análisis ROC se aplicó posteriormente en medicina, radiología, psicología y otras áreas durante varias décadas. Sólo recientemente ha encontrado aplicación en áreas como aprendizaje automático.

Un modelo de clasificación (clasificador) es una función que permite decidir cuáles de un conjunto de instancias están en un grupo o en otro. El resultado del clasificador puede ser un numero real, en el que el límite del clasificador entre cada clase debe determinarse por un valor umbral, o puede ser un resultado discreto que indica directamente una de las clases.

Consideremos un problema de predicción de dos clases o clasificación binaria, en la que los resultados se etiquetan como dos clases: positivos (p) o negativos (n). Hay cuatro posibles resultados a partir de un clasificador binario como el propuesto. Si el resultado de una predicción es p y el valor real es también p, entonces se conoce como un Verdadero Positivo (VP); sin embargo si el valor real es n entonces se conoce como un Falso Positivo (FP). De igual modo, tenemos un Verdadero Negativo (VN) cuando tanto la predicción como el valor real son n, y un Falso Negativo (FN) cuando el resultado de la predicción es n pero el valor real es p.

Se define un experimento a partir de P instancias positivas y N negativas. Los cuatro posibles resultados se pueden formular en una tabla de contingencias 2x2, como se muestra en la figura 6.

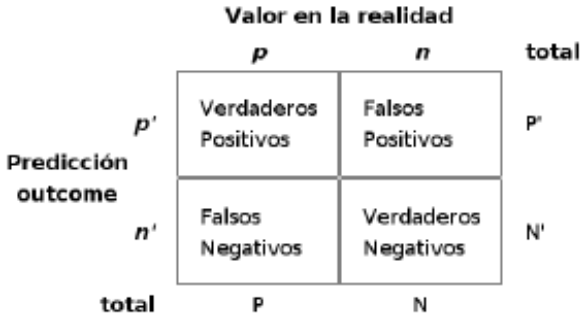

Figura 6 – Tabla de posibles resultados con P instancias positivas y N negativas

2.7.1. El espacio ROC

Para dibujar una curva ROC sólo son necesarias las Relaciones de Verdaderos Positivos (RVP) y de Falsos Positivos (RFP). La RVP mide hasta qué punto un clasificador o prueba diagnóstica es capaz de detectar o clasificar los casos positivos correctamente, de entre todos los casos positivos disponibles durante la prueba. La RFP define cuántos resultados positivos son incorrectos de entre todos los casos negativos disponibles durante la prueba.

Un espacio ROC se define por RFP y RVP como ejes x e y respectivamente, y representa los intercambios entre verdaderos positivos y falsos positivos. Dado que RVP es equivalente a la sensibilidad y RFP es también conocido como (1-especificidad), el gráfico ROC se llama a veces la representación de (1-especificidad) frente a la sensibilidad.

El mejor método posible de predicción se situaría en un punto en la esquina superior izquierda, o coordenada (0,1) del espacio ROC, representando un 100% de sensibilidad (ningún falso negativo) y un 100% también de especificidad (ningún falso positivo). Este punto (0,1) es también llamado una clasificación perfecta. Por el contrario, una clasificación totalmente aleatoria daría un punto a lo largo de la línea diagonal, que se llama también línea de no-discriminación, desde el extremo izquierdo hasta la esquina superior derecha. Un ejemplo típico de adivinación aleatoria sería decidir a partir de los resultados de lanzar una moneda al aire.

2.7.2. Curvas en el espacio ROC

Fijar un valor de umbral determinará un punto en el espacio ROC. Por ejemplo, si se fija ese umbral en 0,8, las probabilidades de las instancias iguales o inferiores serán predichas como positivas, y los valores por debajo serán predichos como negativos. Por tanto se podrá calcular una matriz de confusión para ese umbral de 0,8, y encontrar el punto correspondiente en el espacio ROC. Según se va variando el umbral (por ejemplo, en pasos de 0,1) se tendrá una nueva matriz de confusión y un nuevo punto en el espacio ROC. Dibujar la curva ROC consiste en poner juntos todos los puntos correspondientes a todos los umbrales o puntos de corte, de tal modo que ese conjunto de puntos se parecerá más o menos a una curva en el espacio cuadrado entre (0,0) y (1,1).

La curva ROC, ver figura 7, se puede usar para generar estadísticos que resumen el rendimiento (o la efectividad, en su más amplio sentido) del clasificador. A continuación se proporcionan algunos:

- El punto de intersección de la curva ROC con la línea perpendicular a la línea de no-discriminación.

- El área entre la curva ROC y la línea de no-discriminación.

- El área bajo la curva ROC, llamada comúnmente AUC (*Area Under the Curve*).

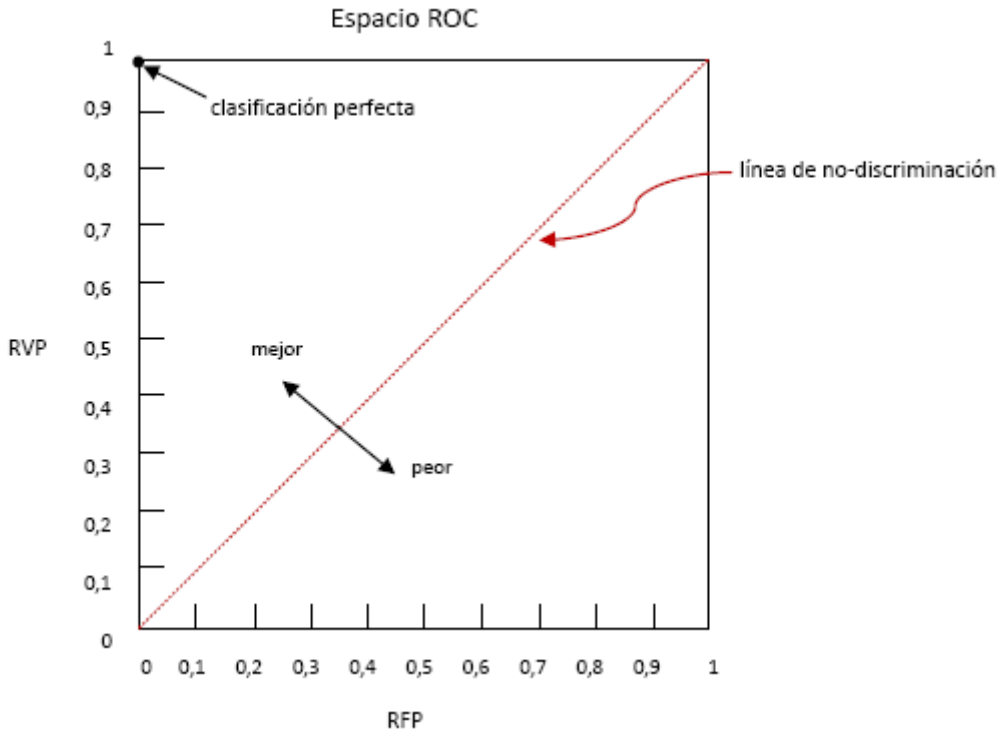

Figura 7 – Ejemplo de curva ROC

El indicador más utilizado en muchos contextos es el área bajo la curva ROC o AUC. Este índice se puede interpretar como la probabilidad de que un clasificador ordenará o puntuará una instancia positiva elegida aleatoriamente más alta que una negativa. En ocasiones puede ser más útil mirar a una región específica de la curva ROC más que a toda la curva. Es posible calcular áreas parciales bajo la curva, o AUC parciales. Por ejemplo, nos podríamos concentrar en la región de la curva con razones de falsos positivos más bajas.

Un resumen de la terminología más utilizada para la representación y análisis de las curvas ROC sería la siguiente:

- Verdaderos Positivos (VP): éxitos
- Verdaderos Negativos (VN): rechazos correctos
- Falsos Positivos (FP): falsas alarmas

- Falsos Negativos (FN): no detección

- Razón de Verdaderos Positivos (RVP): RVP= VP/V = VP/(VP+FN)

- Razón de Falsos Positivos (RFP): RFP=FP/N=FP/(FP+VN)

2.8. Motores gráficos

Bajo este concepto se engloban las librerías o componentes informáticos que generan imágenes a partir de modelos descritos en un lenguaje o estructuras de datos [21]. Junto a motores físicos, los motores gráficos forman los llamados motores de juegos. A continuación se enumeran algunos de los motores gráficos más utilizados.

- OpenGL (Open Graphics Library). Desarrollado por Silicon Graphics desde 1992. Es el motor de infinidad de entornos de desarrollo CAD, visualizadores de información científica y administrativa, y de juegos y simuladores en dos y tres dimensiones. La versión actual es la 4.1.

- DirectX (DirectDraw y Direct3D). DirectDraw y Direct3D son los componentes 2D y 3D, respectivamente, de DirectX. DirectX es el paquete de desarrollo software de Microsoft, en desarrollo desde 1995, para el control de bajo nivel de periféricos y dispositivos. Su versión actual es la 11. Es el principal rival de OpenGL, con la contrapartida de solo ser compatible con sistemas operativos de Microsoft.

- Irrlicht, OGRE,... Son muchas las librerías de renderización 3D implementadas sobre OpenGL y DirectX que proporcionan una API única para el empleo de cualquiera de las dos opciones.

- GTK, Qt, FLTK,... Existen multitud de librerías multiplataforma que integran llamadas a las APIs nativas de los sistemas operativos objetivo.

2.8.1. GTK+

GTK+ o The GIMP Toolkit es un conjunto de librerías multiplataforma para desarrollar interfaces gráficas de usuario (GUI), principalmente para los entornos gráficos GNOME, XFCE y ROX aunque también se puede usar en el escritorio de Windows, MacOS y otros.

Inicialmente fueron creadas para desarrollar el programa de edición de imagen GIMP, sin embargo actualmente se usan bastante por muchos programas en los sistemas GNU/Linux. Junto a Qt es una de las librerías más populares para X Window System.

GTK+ se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Ruby, Perl, PHP o Python. Licenciado bajo los términos de LGPL, GTK+ es software libre y es parte del proyecto GNU.

2.8.2. Librerías de GTK+

GTK+ se basa en varias librerías del equipo de GTK+ y de GNOME:

- Glib. Librería de bajo nivel estructura básica de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución como cicles, hilos, carga dinámica o un sistema de objetos.

- GTK. Librería la cual realmente contiene los objetos y funciones para crear la interfaz de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.

- GDK. Librería que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.

- ATK. Librería para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidos. Pueden usarse utilidades como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.

- Pango. Librería para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.

- Cairo. Librería de renderizado avanzado de controles de aplicación.

Capítulo 3

Desarrollo

3.1. Descripción detallada

Como se ha mencionado en las secciones anteriores el presente estudio está integrado dentro del desarrollo del prototipo del robot asistencial ASIBOT. Un robot, desarrollado por el grupo Robotics Lab de la Universidad Carlos III de Madrid, cuya finalidad es la ayuda y asistencia a personas discapacitadas y de la tercera edad.

El objetivo de este proyecto es la mejora de las características funcionales del robot y facilitar su utilización al usuario. Estas mejoras consisten en la integración de un sistema de visión por computador que permita detectar objetos, mostrando su ubicación en una pantalla. Se pensó en realizar el estudio sobre un bote de Coca-Cola, siendo éste un objeto cotidiano y de uso general.

Para alcanzar los objetivos propuestos se propuso diseñar una aplicación informática ejecutable bajo entorno Linux que muestre la imagen de una cámara de red (cámara IP), a elegir por el usuario entre varias disponibles. Mostrada en la pantalla de un computador la imagen captada por la cámara, el programa informático debe ser capaz de detectar la presencia, dentro del campo de visión de la cámara, de un bote de Cola-Cola. En caso de que exista, éste se marcará con un cuadrado y se mostrarán sus coordenadas espaciales en píxeles dentro de la pantalla. Para ello se realizará un entrenamiento mediante el Adaboost (ver sección 2.5.3 y en concreto el utilizado en este proyecto es el correspondiente al apartado 2.5.3.3) y se obtendrá un archivo xml que se podrá integrar en el código de la aplicación utilizando funciones de las librerías OpenCV (ver sección 2.4), dirigida fundamentalmente a la visión por computador en tiempo real.

Una vez conseguido que la aplicación informática funcione correctamente, se debe diseñar una interfaz gráfica e implementar el programa en ella. La función de dicha interfaz es facilitar la utilización del programa al usuario, mejorar su aspecto visual y poder seleccionar la cámara deseada entre las disponibles. Para su diseño nos basaremos en la utilización de GTK+ (ver apartado 2.8).

3.2. Diseño del sistema

Para el desarrollo de este proyecto se ha seguido un método de desarrollo en cascada ascendente y descendente. Es decir, se permite la posibilidad de, cuando se han encontrado problemas en una fase, volver a la fase anterior para realizar cambios.

Se puede estructurar el estudio en dos partes: primero, crear la aplicación informática detector del bote de Coca-Cola; y segundo, diseñar la interfaz gráfica implementando el programa creado en la parte primera.

Los pasos a seguir para conseguir el resultado final son los siguientes:

- Crear la aplicación detector del bote de Coca-Cola:

    1. Mostrar la imagen en tiempo real de la cámara de red en una ventana.

    2. Generar el archivo xml del clasificador mediante entrenamiento.

    3. Realizar test y obtener las gráficas de cada entrenamiento y hacer una comparación para elegir el mejor.

    4. Crear la aplicación informática que muestre la imagen de la cámara de red en una ventana y detecte el bote de Coca-Cola.

- Diseñar una interfaz gráfica para la aplicación:

5. Implementar la aplicación de OpenCV con GTK+.

6. Caracterizar la interfaz gráfica.

7. Insertar las coordenadas del cuadro de detección en la pantalla.

A continuación se va a explicar detalladamente cada paso seguido en el desarrollo del proyecto, desde el inicio hasta conseguir la aplicación final objeto del estudio.

3.2.1. Captura de imágenes en tiempo real

Para empezar a trabajar en el desarrollo del proyecto y familiarizarnos con la utilización de las librerías de OpenCV se va a crear una aplicación, básica y sencilla, en la que se muestra una ventana con la imagen en tiempo real capturada por una cámara de red.

Al programa que implementa esta funcionalidad se le ha llamado “Programa 1”. Su código se incluye en el Apéndice A.II.

El resultado de la ejecución de la aplicación se muestra en la figura 8.

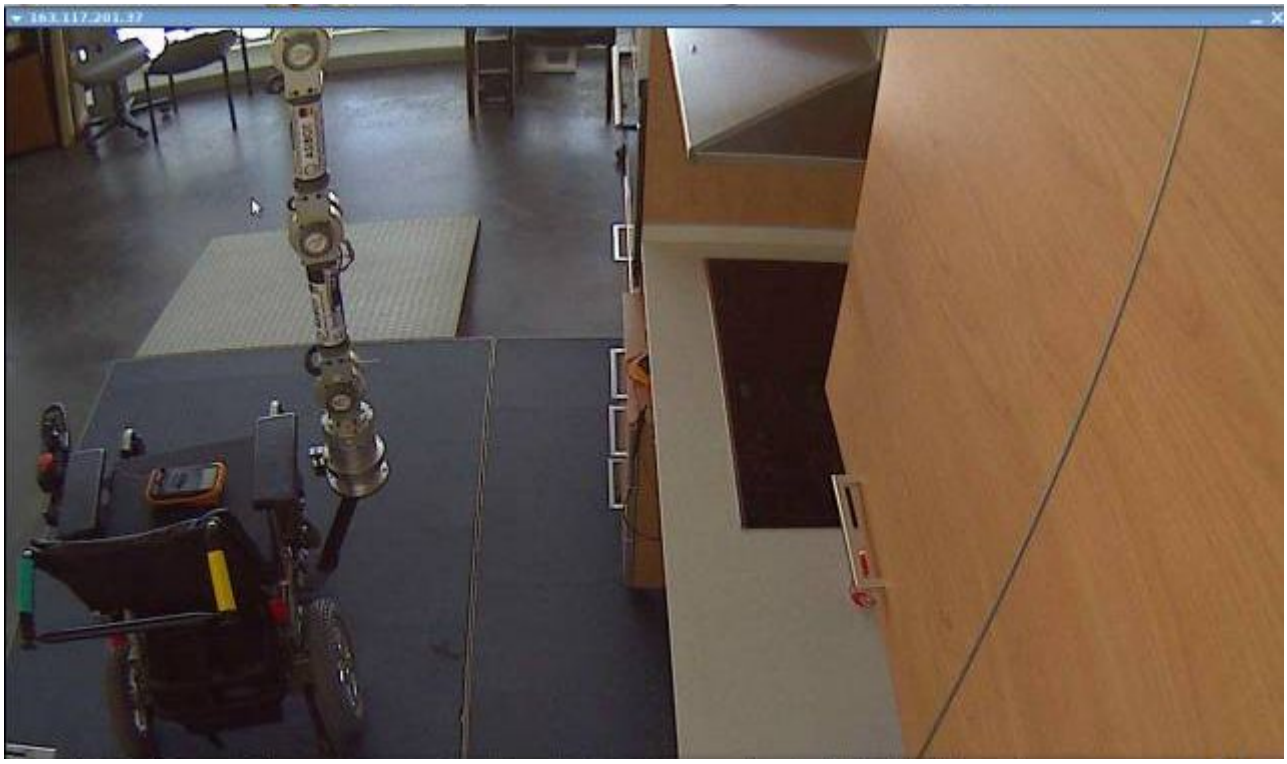

Figura 8 – Resultado de ejecución del “Programa 1”

Si se quisiera realizar un programa para visualizar otra cámara de red simplemente habría que cambiar la dirección IP de la cámara actual por la nueva.

Por ejemplo, en la línea 8 del código del “Programa 1” cambiaríamos:

```
CvCapture* capture = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");
```

Por:

```
CvCapture* capture = cvCreateFileCapture("http://root:root@163.117.201.38/mjpg/video.mjpg");
```

El resultado sería el mostrado en la figura 9.

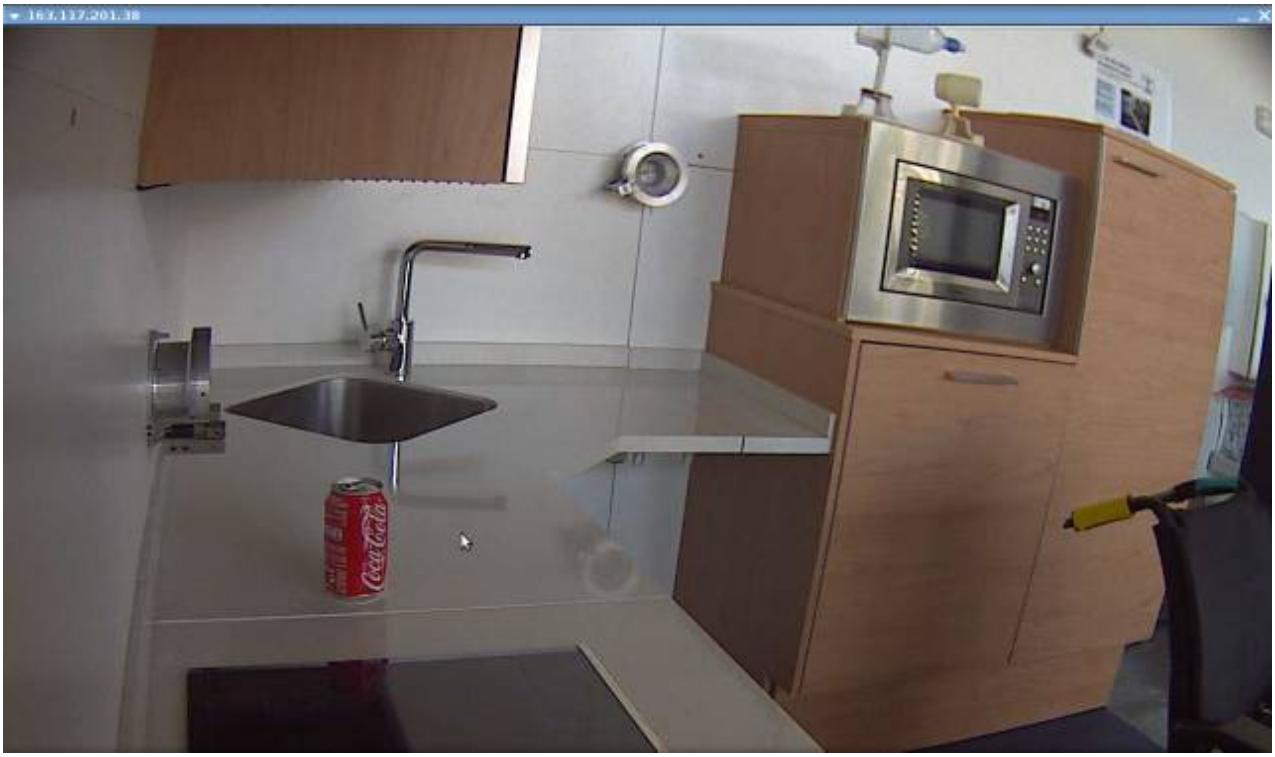

Figura 9 – Resultado de ejecución del "Programa 1" con otra cámara

3.2.2. Generación del clasificador

La obtención de un clasificador (detector de objetos) en árbol construido mediante el método de Boosting (ver sección 2.5.6) para un objeto en concreto requiere el seguimiento estricto de unos pasos que se van a describir con detalle en este capítulo [22]. Una vez completados dispondremos de un potente y rápido sistema que nos permitirá procesar imágenes a una muy elevada velocidad para determinar si existe o no un bote de Coca-Cola en cada imagen y en caso afirmativo conocer su situación exacta dentro de ella.

En primer lugar, se requerirá una preparación exhaustiva de las imágenes que servirán como muestras, tanto positivas como negativas, para el aprendizaje del clasificador. En un caso ideal, el algoritmo partirá del total del espacio muestral, distribuido en muestras positivas y negativas. Así el clasificador obtenido tendrá una probabilidad de detección máxima y la probabilidad de falsa alarma mínima. Sin embargo, por muy elevado que sea el conjunto de muestras facilitado al algoritmo, nunca podremos

representar la totalidad del espacio muestral; motivo por el cual resulta fundamental proporcionar muestras bien seleccionadas, que representen de la mejor forma posible al total. Por tanto, este primer paso del proceso, la *Preparación de las imágenes de muestra*, es la base para obtener un clasificador con buena fiabilidad en la detección.

Será necesario distinguir en esta parte, la preparación de las muestras positivas y de las negativas, ya que en el primer caso, el proceso se ha de centrar en la calidad de las mismas y en el segundo en la variedad.

Seguidamente, se pasará a la parte de *Entrenamiento del clasificador* en árbol mediante el método del Boosting. Esta parte del proceso es la realizada por el programa llamado Haartraining (ver apartado 2.5), la elección de los parámetros de entrenamiento, tales como el tipo de características, el tamaño de la muestra, la cantidad de ejemplos positivos y negativos, entre otros, será también motivo de estudio y pruebas. Al finalizar el entrenamiento, se obtendrán como resultado las diferentes etapas del clasificador.

En tercer lugar, se procederá a la comprobación del clasificador obtenido utilizando el programa llamado Performance [23]. Durante este periodo de *Uso y Pruebas del clasificador* analizaremos las curvas ROC (ver sección 2.7), una gráfica de trama de la sensibilidad, o la tasa de verdaderos positivos frente a la tasa de falsos positivos, que nos servirán para extraer las conclusiones sobre las calidades de los clasificadores y poder elegir el de mejor funcionamiento.

3.2.2.1. Preparación de las imágenes de muestra

Esta primera fase del proceso ha ido sufriendo numerosos cambios desde el inicio de este proyecto, todos ellos motivados por la experiencia y los resultados obtenidos de los clasificadores que se iban generando. Además, se ha podido comprobar a lo largo de todas las pruebas realizadas en todo el periodo de estudio, la gran importancia de esta primera etapa preparatoria, ya que de ella depende en gran medida la calidad y fiabilidad del clasificador obtenido con cada entrenamiento.

El método de entrenamiento debe encontrar las características que mejor clasifiquen al objeto buscado, así como en combinarlas de la mejor forma posible para conseguir obtener un detector de objetos en forma de árbol muy potente. Esta etapa preparatoria será precisamente la clave del éxito para encontrar las mejores características, por tanto, se requiere aplicar una excesiva meticulosidad a la hora de seleccionar las imágenes que harán las veces de muestras positivas y de muestras negativas.

La preparación de las imágenes para entrenamiento consiste básicamente en proporcionar al Haartraining (ver sección 2.5) dos ficheros; el primero de ellos contendrá una lista de imágenes en las que no se encuentre el objeto a detectar, que constituirán las muestras negativas. Las imágenes se deben proporcionar con su ruta de acceso completa (directorio y nombre de la imagen). Pero donde reside la importancia de este proceso es en la preparación de las imágenes que se encontrarán en esa lista.

Y el otro fichero proporcionado al Haartraining para el entrenamiento será el que contendrá las muestras positivas. Este archivo será de nuevo una lista de imágenes, pero acompañadas además del número de muestras positivas (objetos a detectar) que contiene la imagen y de las coordenadas en las que se encuentran dichas muestras dentro de ella en formato x_1 y_1 Δx Δy, tantas veces como objetos indique el numero antes indicado, donde (x_1 , y_1) será el punto correspondiente a la esquina superior izquierda de la muestra, Δx y Δy será respectivamente el ancho y el alto de la muestra en esa imagen, partiendo desde el punto anterior.

Nos centraremos en la preparación de las imágenes que formarán las dos listas para ambos ficheros explicando su elaboración.

3.2.2.1.1. Muestras negativas

Las imágenes negativas son aquellas tomadas de forma arbitraria. Lo más importante de estas imágenes, es que no contengan el objeto a clasificar, en este caso un bote de Coca-Cola. Estas imágenes pueden ser de diferentes lugares.

Las muestras negativas requieren un fichero de índices de extensión txt para que el programa pueda trabajar con ellas. Este fichero contiene el nombre de las imágenes usadas como fondos con su dirección relativa dentro del sistema de ficheros.

Una pequeña muestra del contenido de este archivo es la mostrada en la figura 10.

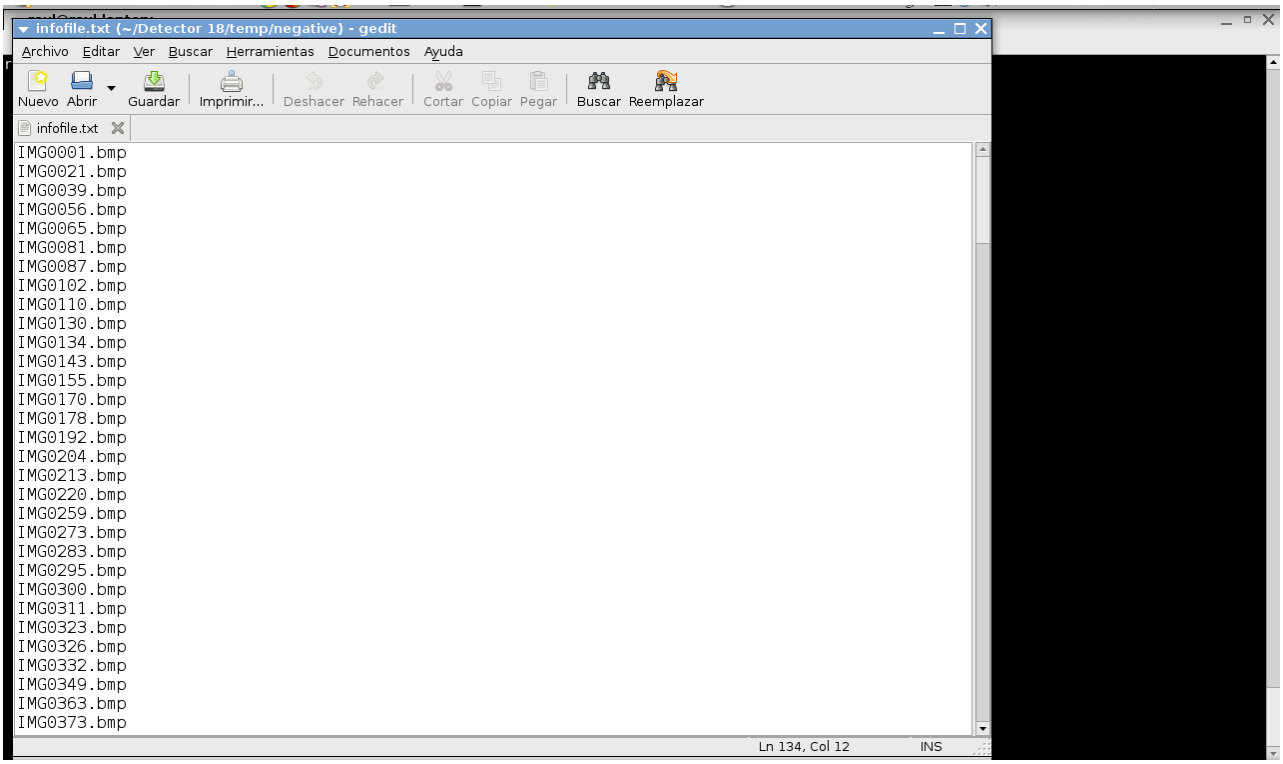

Figura 10 – Ejemplo del contenido del archivo de muestras negativas

Se comenzó por averiguar exactamente el proceso que realiza el Haartraining con las imágenes de fondo para encontrar candidatos negativos. El sistema utiliza las imágenes que se le facilitan para buscar dentro de ellas subventanas del tamaño deseado para la muestra positiva que aporten información para la construcción del clasificador.

En cuanto ha barrido todas las imágenes facilitadas en busca de candidatos válidos para el aprendizaje, vuelve a empezar por la primera. Este sistema se repite durante todo el entrenamiento hasta que se finaliza la construcción del clasificador.

El contenido de esas imágenes debe estar suficientemente cuidado como para que sea factible sacar de ellas información para el aprendizaje.

En la figura 11 se puede observar una pequeña muestra de imágenes negativas utilizadas durante el desarrollo del proyecto (caracterizadas por la ausencia del bote de Coca-Cola):

Figura 11 – Ejemplos de imágenes negativas

3.2.2.1.2. Muestras positivas

Son imágenes en las que se encuentra el objeto a clasificar de una forma clara, en este caso imágenes de un bote de Coca-Cola.

Para crear el fichero de muestras positivas, se utiliza un programa denominado Objectmarker [24], que es un programa realizado en C++ y que nos va a permitir señalar la ubicación de los botes de Coca-Cola en cada una de las muestras.

El programa toma como entrada un fichero que contenga una lista de imágenes a visualizar (las imágenes deben estar en formato .bmp), lo abre y va leyendo línea a línea. Representa por pantalla la imagen referenciada por cada línea del fichero. Una vez representada la primera de las imágenes, el programa permanece en estado de espera hasta la llegada de algún evento de teclado o de ratón para los que está preparado para responder. Los eventos de teclado y las acciones que provocan en el programa son los siguientes:

<*Enter*>: Guarda los rectángulos añadidos y muestra la siguiente imagen
<*ESC*>: Sale del programa
<*Espacio*>: Añade las coordenadas del rectángulo en la imagen actual

El objetivo más importante del Objectmarker es precisamente seleccionar (recortar) los objetos y crear un fichero de índices con información sobre el conjunto de las muestras para el entrenamiento. Este archivo de índices (de extensión txt) tendrá la siguiente estructura:

nombre_archivo1 *número_muestras* x_{11} y_{11} w_{11} h_{11} x_{12} y_{12}...
nombre_archivo2 *número_muestras* x_{21} y_{21} w_{21} h_{21} x_{22} y_{22}...
...

Siendo *nombre_archivo* el nombre de la imagen donde se encuentran los objetos a buscar. Debemos destacar que el nombre del archivo debe ir acompañado de su dirección absoluta dentro del sistema de ficheros, *número_muestras* es el número de objetos positivos que se

encuentran en la imagen, en nuestro caso el número de botes de Coca-Cola que contiene; x_{II} e y_{II} las coordenadas del primer punto del rectángulo, w_{II} y h_{II} el ancho y alto del rectángulo dibujado que contiene el bote; y así tantos parámetros *x*, *y*, *w* y *h* como botes haya en la imagen.

El proceso es el siguiente:

Para una determinada imagen, se hace "click" con el ratón sobre un determinado punto y se arrastra. Entonces se irá dibujando un rectángulo con el que podremos englobar el objeto que deseemos. Al volver a hacer "click", fijaremos el tamaño de la caja. Si entonces pulsamos la tecla "espacio", se almacenará en el fichero que se haya especificado en la entrada como fichero destino, la imagen con las coordenadas de los dos puntos de la imagen donde se hizo "click".

En la figura 12 se puede observar el contenido del archivo de índices de las imágenes positivas.

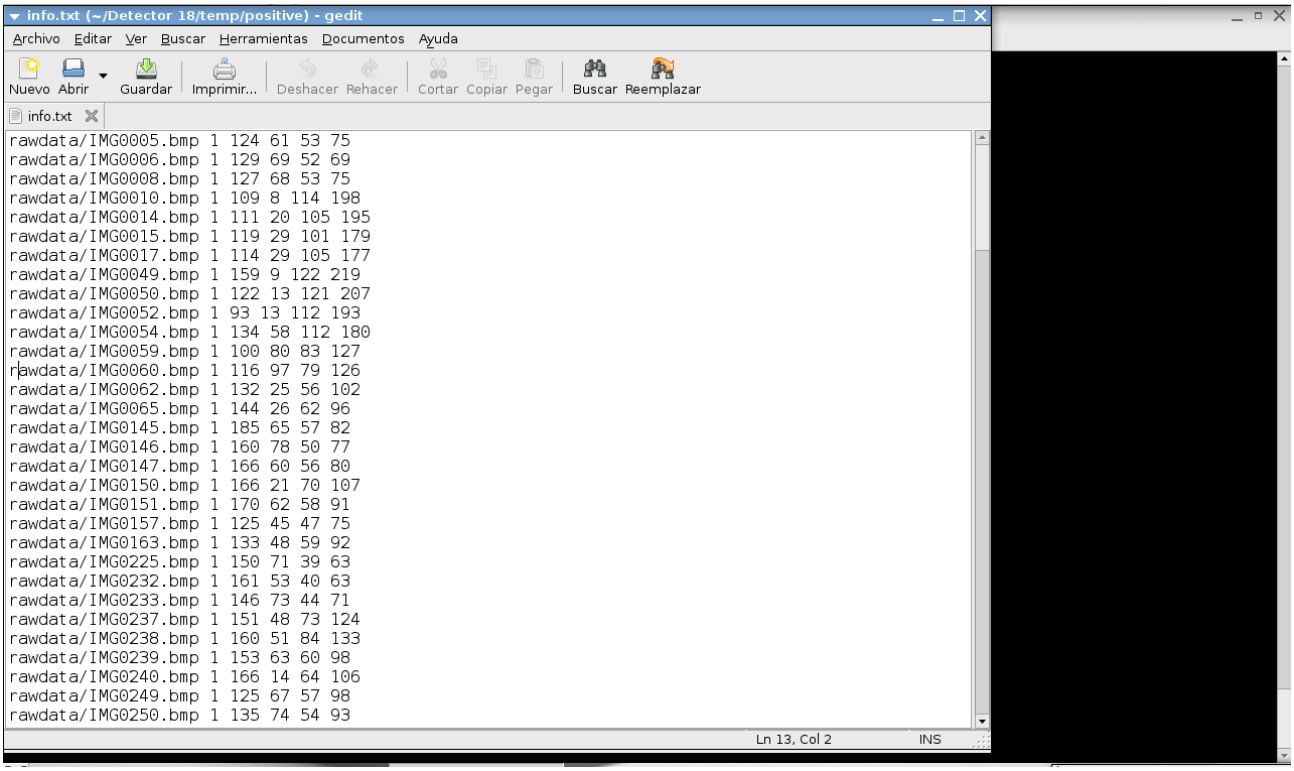

Figura 12 – Muestra del archivo de imágenes positivas

Las imágenes positivas utilizadas para el desarrollo de este proyecto han sido tomadas desde diferentes ángulos, con diferentes fondos y diferente iluminación. En la figura 13 se puede observar una pequeña muestra de estas imágenes:

Figura 13 – Ejemplos de imágenes positivas

Así disponemos de los ficheros preparados para que sirva como entrada a la aplicación Createsamples [25], el cual generará otro archivo con un formato determinado preparado para su lectura por el Haartraining.

3.2.2.2. Entrenamiento del clasificador

Una vez adquiridas las imágenes positivas y negativas y generados los archivos correspondientes se pasará a la siguiente etapa, crear la muestra. Para ello se utilizará el programa Createsamples. La llamada al programa desde el intérprete de comandos para crear el fichero .vec ha sido, durante el transcurso de la totalidad del proyecto la siguiente:

```
opencv-createsamples -info fichero muestras -vec fichero.vec -num número muestras  -w ancho -h alto
```

- *info*: Precederá al nombre del fichero lista que contendrá todas las imágenes de los objetos que constituyen las muestras positivas y cuya construcción se ha descrito en la sección anterior.

- *vec*: Tras esta etiqueta escribiremos el nombre del fichero .vec que creará el Createsamples a partir de las muestras proporcionadas.

- *num*: Indica al Createsamples el número de muestras que contiene el fichero lista y que por tanto añadirá en el formato adecuado en el fichero .vec.

- *w*: Indica el ancho deseado para las muestras creadas en la ejecución del Createsamples. Su valor por defecto es de 24 píxeles.

- *h*: Indica el valor para el alto de las muestras. Su valor por defecto es de 24 píxeles.

Con la llamada al programa especificada arriba, donde se introducen como parámetros de entrada al Createsamples el fichero con la lista y el fichero destino donde almacenará las muestras con el tamaño deseado, el programa ejecuta una función llamada cvCreateTrainingSamplesFromInfo cuyo objetivo es redimensionar las muestras al tamaño solicitado y almacenarlas en el fichero destino .vec.

Para la elaboración de nuestro clasificador se crearon las muestras con un tamaño de 50x65 píxeles.

En concreto la llamada a la aplicación para nuestro detector del bote de Coca-Cola es la siguiente:

```
opencv-createsamples -info positive/info.txt -vec data/vector.vec
-num 165 –w 50 -h 65
```

Tras realizar este proceso se creará un vector que contiene las imágenes de muestra. Una vez preparadas las imágenes, ya se puede iniciar el entrenamiento de un clasificador mediante el método de entrenamiento. Para ello se utilizó el programa construido por Intel llamado Haartraining (ver sección 2.5).

Un clasificador es realmente un conjunto de directorios, uno por etapa, que contienen un archivo de texto donde se han escrito las características seleccionadas en un formato muy similar al que hemos visto para construirlas.

En la figura 14 podemos observar el contenido real del fichero AdaBoostCARTHaarClassifier.txt que constituye la primera etapa de un clasificador para botes de Coca-Cola construido con 165 muestras positivas y 200 candidatos negativos [26]. En este fichero se encuentra toda la información necesaria para decidir si una subventana es el objeto buscado o no. La primera cifra del fichero indica el número de clasificadores débiles en la etapa. El siguiente es la cantidad de nodos que tiene el árbol de clasificación. El siguiente refleja el número de rectángulos de los que se compone el nodo, seguido de las mismas líneas, una por cada rectángulo especificando sus coordenadas que se encuentran ordenadas en la forma: x_{origen} , y_{origen} , *ancho, alto, 0, peso*. La siguiente línea recoge el nombre de la característica en cuestión y la siguiente indica el umbral de rama (si la respuesta <= umbral el camino a seguir es la rama izquierda, si no el camino es la rama derecha) y los índices de rama izquierda y derecha respectivamente. Se muestran los parámetros de todos los nodos y finalmente se muestran las tasas de falsa alarma del nodo. Todos los valores anteriores se repiten para todos los clasificadores que forman la etapa.

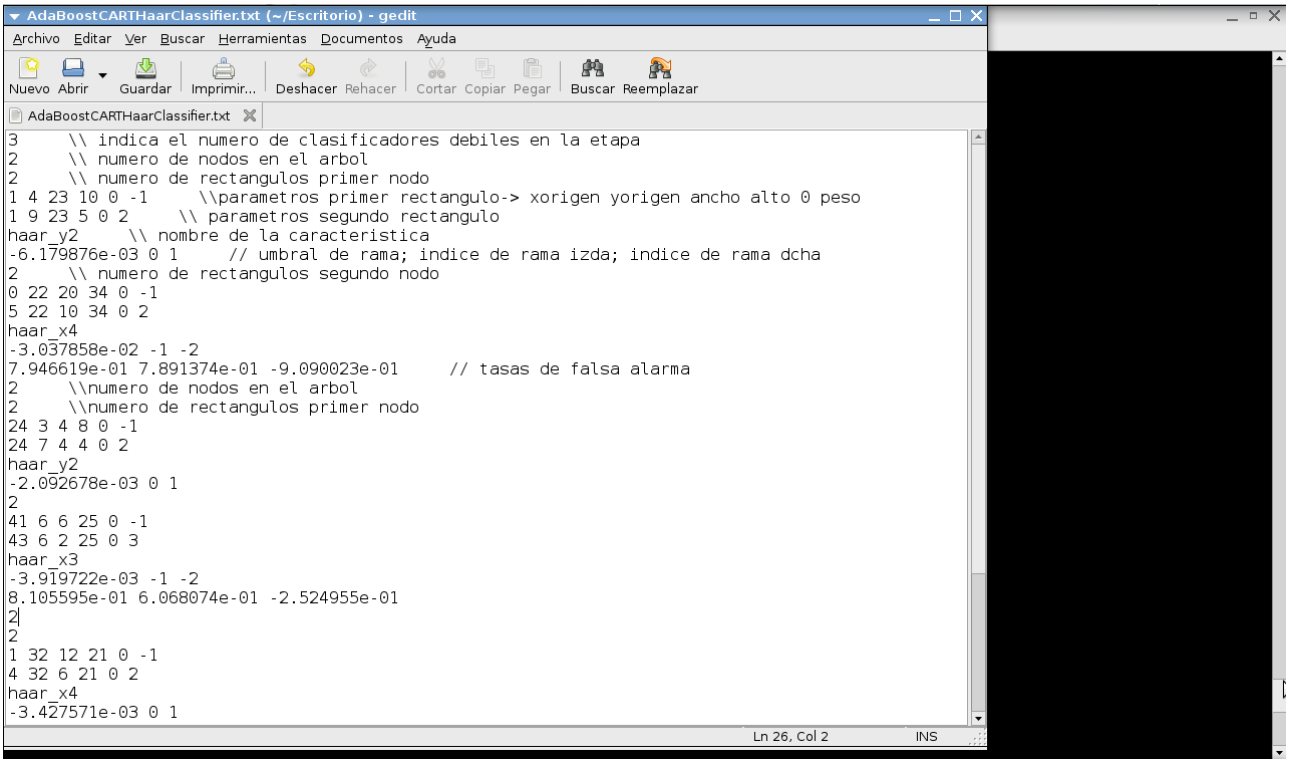

Figura 14 – Contenido del fichero de la primera etapa del clasificador.

La llamada a la función para entrenar el detector del bote de Coca-Cola es la siguiente:

```
opencv-haartraining -data data/cascade -vec data/vector.vec -bg negative/infofile.txt -npos 165 -nneg 200 -nstages 30 -mem 1000 -mode ALL -w 50   -h 65 -nsplits 2 -minhitrate 0,999
```

El significado de cada uno de los argumentos:

- *data*: Se utiliza para especificar el directorio donde el programa irá almacenando el clasificador a medida que lo construya. Haartraining crea un directorio para cada una de las etapas con el nombre del número de la etapa, donde almacena el archivo AdaBoostCARTHaarClassifier.txt cuya estructura hemos visto en el apartado anterior.

- *vec*: Tras esta etiqueta se especifica el fichero con extensión .vec creado mediante Createsamples, que contiene la información de las muestras positivas.

- *bg*: Precede al fichero que contenga la lista de imágenes de fondo en las que se buscarán los candidatos negativos.

- *npos*: Se introducen tras esta etiqueta el número de muestras positivas que contiene el fichero .vec. Su valor por defecto es 2000.

- *nneg*: Aquí se introduce el número de muestras negativas que se le solicita al entrenamiento que encuentre en las imágenes de fondo del tamaño especificado. Su valor por defecto también es 2000.

- *nstages*: Sirve para especificar el número de etapas deseadas de las que se compondrá el clasificador. El valor por defecto es 14.

- *mem*: Selecciona la memoria que se va a utilizar para el proceso de entrenamiento.

- *mode*: Permite escoger para las características, si se desea el conjunto básico (BASIC) o el conjunto extendido (ALL). La forma por defecto es la primera.

- *w*: Sirve para especificar el ancho de las muestras positivas proporcionadas. Su valor por defecto es 24 píxeles.

- *h*: Sirve para especificar el alto de las muestras positivas. Su valor por defecto es también 24 píxeles.

- *nsplits*: Hace referencia a la estructura del árbol de clasificación, que determina la debilidad del clasificador. En nuestro caso se utilizará el clasificador con 2 divisiones internas de nodos.

- *minhitrate*: Mínima tasa de detección que se desea alcanzar en cada etapa, siendo la tasa del clasificador completo ese mismo valor elevado al número de etapas. El valor por defecto es 0,995.

Durante todo el transcurso del proyecto se estuvieron realizando entrenamientos de diferentes clasificadores investigando sobre los efectos del proceso de preparación de las imágenes y los valores para los parámetros de entrada.

Para preparar las muestras positivas es necesario recortar una a una mediante el Objectmarker para después realizar todo el proceso que conduce a la creación del fichero .vec mediante el Createsamples. En general para el entrenamiento de un clasificador se deben reunir más de 1000 muestras, pero para el detector del bote de Coca-Cola y tras el estudio y las pruebas realizadas se verá, que eligiendo correctamente las muestras, con menos de 200 el detector tiene mejor calidad. En concreto para la construcción del clasificador de este proyecto se han utilizado 165 muestras positivas.

En el caso de las muestras negativas, el archivo que se utiliza contiene imágenes completas, así que el programa puede barrer cada una de las imágenes del archivo y encontrar varios candidatos negativos. De este modo, el archivo puede contener 400 imágenes por ejemplo, y el flag -nneg puede valer 2000. Sin embargo, la mejor práctica será ajustar el valor de -nneg al número de imágenes reales que proporcione el fichero de fondos. En este estudio se han tomado 200 imágenes como muestras negativas.

En cuanto al número de etapas del clasificador, se podría decir que hay un número óptimo diferente para cada clasificador dependiendo del resto de parámetros y del tipo de imágenes y muestras a las que vaya destinado. Pero por lo general, se puede asegurar que un clasificador presentará un buen comportamiento a partir de las 14 etapas que efectivamente es el valor por defecto del programa.

El resto de parámetros dependen totalmente del tipo de prueba que se desee realizar.

El Haartraining genera un archivo xml cuando el proceso haya terminado. Si se desea convertir los archivos de salida del Haartraining en un archivo xml, si éste aun no ha concluido con el entrenamiento, se debe usar la aplicación cascade2xml.

La llamada a la aplicación cascade2xml para nuestro detector del bote de Coca-Cola sería la siguiente:

```
./cascade2xml.exe data cascada.xml 50 65
```

Donde,

- *data*: Indica el directorio donde se han almacenado los archivos del clasificador.

- *cascada.xml*: Se especifica el archivo con extensión .xml creado por el cascade2xml, que contiene la información de los archivos del clasificador.

- *50 65*: Es el ancho y alto, respectivamente, de las muestras utilizadas durante el entrenamiento con el Haartraining.

3.2.3. Integración del clasificador en aplicación C++

Para comenzar con el diseño de la aplicación informática del detector se va a realizar un programa ejecutable que muestre la imagen de una cámara de red (cámara IP), que cargue el archivo xml creado en el entrenamiento del clasificador del detector, que detecte la presencia de un bote de Coca-Cola y que el resultado lo muestre por pantalla. Si hay bote, éste se marcara con un rectángulo.

Al programa que implementa esta funcionalidad se le ha llamado “Programa 2”. Su código se incluye en el Apéndice A.III.

En resumen, los pasos seguidos para el diseño de esta aplicación son los siguientes:

Función principal (main):

1. Crear un puntero a imagen y otro a copia de imagen.

```
IplImage *frame, *framecopy = 0;
```

2. Cargar clasificador según xml creado en el entrenamiento.

```
classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);
```

3. Crear memoria de almacenamiento de cálculos.

```
MemStorage = cvCreateMemStorage(0);
```

4. Tomar fuente de entrada, en este caso la dirección IP de la cámara de red.

```
CvCapture* inputSource = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");
```

5. Verificar fuente de entrada y clasificador.

```
if(!inputSource || !classifier)
  return -1;
```

6. Crear ventana para mostrar el resultado.

```
cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);
```

7. Grabar y almacenar la imagen del video.

```
frame = cvRetrieveFrame(inputSource);
```

8. Crear copia modificada de la imagen.

```
      if(!framecopy)
         framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);
```

9. Si el origen de la imagen está en la parte superior izda. se copia la imagen en el puntero a copia de imagen (framecopy), en caso contrario (origen en la parte inferior izda.) la voltea alrededor del eje x.

```
if(frame->origin == IPL_ORIGIN_TL)
  cvCopy(frame, framecopy, 0);

else
  cvFlip(frame, framecopy, 0);
```

10. Llamar a la función Detection.

```
Detection(framecopy);
```

11. Liberar copia de imagen y fuente de entrada.

```
cvReleaseImage( &framecopy );
cvReleaseCapture( &inputSource );
```

Función Detection:

Para poder detectar mejor lo primero que hay que hacer es ecualizar el histograma (ver sección 2.3.4) de la imagen tomada y para ello necesitamos pasar la imagen a un tamaño determinado y a escala de grises. Para ello hacemos los siguientes pasos:

1. Crear una imagen (que pasará a escala de grises).

```
gray = cvCreateImage( cvSize(img->width, img->height), 8, 1);
```

2. Crear otra imagen a la escala que se le indique (mayor que 1), con lo que la imagen queda más pequeña.

```
small_img = cvCreateImage( cvSize( cvRound (img->width/escala), cvRound (img->height/escala)), 8, 1);
```

3. Convertir la primera imagen creada a escala de grises.

```
cvCvtColor( img, gray, CV_BGR2GRAY);
```

4. Cambiar el tamaño de la imagen a escala de grises igual a la imagen a escala.

```
cvResize( gray, small_img, CV_INTER_LINEAR);
```

5. Realizar la ecualización del histograma.

```
cvEqualizeHist( small_img, small_img);
```

6. Limpiar memoria de almacenamiento de cálculos.

```
cvClearMemStorage(MemStorage);
```

Si se ha creado bien el clasificador:

7. Crear puntero a los rectángulos que identifican a los botes en la imagen modificada y a escala de grises.

```
CvSeq* bote = cvHaarDetectObjects(small_img, classifier, MemStorage, 1.2, 2, 0, cvSize(50, 65));
```

Los parámetros de esta función son:

bote: Nombre del puntero.
small_img: Imagen.
classifier: Clasificador.
MemStorage: Memoria reservada para las imágenes con rectángulo.
1.2: Escala.
2: Número mínimo (menos 1) de rectángulos vecinos que quedan encima de un objeto.
0: Modo de operación.
cvSize(50,65): Tamaño en píxeles de las imágenes muestras.

Dentro de esta imagen, para cada bote detectado:

7.1. Obtener las características de uno de los rectángulos.

```
CvRect* rectangle = (CvRect*)cvGetSeqElem(bote, i);
```

7.2. Obtener las coordenadas de los dos puntos que definen este rectángulo (teniendo en cuenta la escala).

```
point1.x = cvRound((rectangle->x)*escala);
point2.x = cvRound((rectangle->x + rectangle->width)*escala);
point1.y = cvRound((rectangle->y)*escala);
point2.y = cvRound((rectangle->y + rectangle->height)*escala);
```

7.3. Dibujar el rectángulo en la imagen.

```
cvRectangle( img, point1, point2, CV_RGB(255,0,0), 3, 8, 0 );
```

Los parámetros de la función son:

img: Imagen.
point1: Punto 1 del rectángulo, corresponde al vértice del rectángulo.

point2: Punto 2 del rectángulo, vértice opuesto al punto 1.

CV_RGB(255,0,0): Define el color de las líneas del rectángulo.

3: Define el grosor de las líneas del rectángulo.

8: Tipo de líneas del rectángulo.

0: Número de bits fraccionarios en las coordenadas de los puntos.

8. Mostrar imagen en ventana.

```
cvShowImage( "Result", img);
```

9. Liberar imagen a escala de grises e imagen modificada en escala de grises.

```
cvReleaseImage( &gray);
cvReleaseImage( &small_img);
```

El resultado de la ejecución de esta aplicación se puede observar en la figura 15.

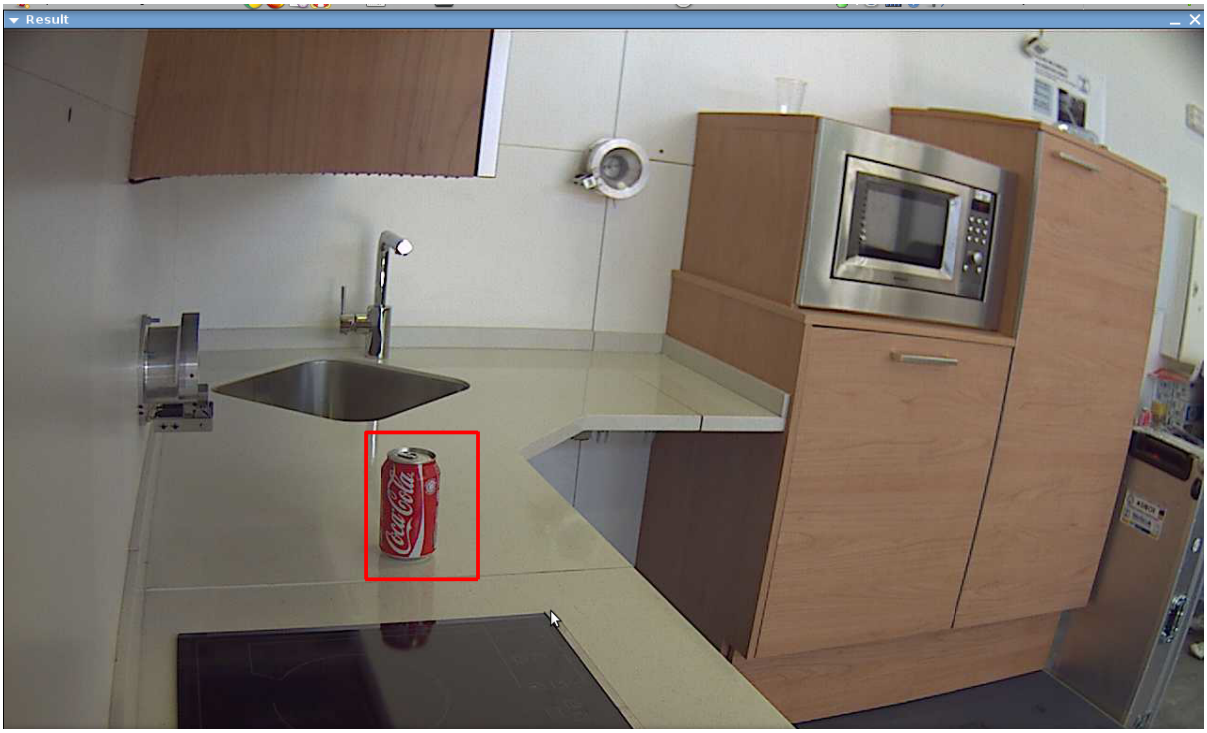

Figura 15 – Imagen del resultado de ejecución del "Programa 2"

3.2.4. Implementación del módulo de usuario

Una vez realizado el programa para mostrar la imagen de la cámara de red, cargar el clasificador para detectar la presencia del bote de Coca-Cola y en caso afirmativo marcarlo mediante un rectángulo vamos a diseñar una interfaz gráfica para facilitar su manejo al usuario y mejorar su funcionalidad. Para ello primero se creará un programa sencillo para familiarizarse con la librería GTK+, para después diseñar el programa final implementando el "Programa 2", código incluido en el Apéndice A.III, con la interfaz gráfica.

3.2.4.1. Crear ventana con GTK+ con un botón y una función

Antes de diseñar la aplicación final objeto del proyecto y para empezar a familiarizarse con el uso de las librerías de GTK+ se creará un programa básico. Al ejecutar dicho programa se mostrará una ventana con un botón y un texto. Si se presiona en el botón se invierte la frase mostrada y así sucesivamente. Para salir del programa presionamos sobre el aspa en la parte superior derecha de la ventana.

Al programa que implementa esta funcionalidad se le ha llamado "Programa 3". Su código se incluye en el Apéndice A.IV.

El programa consta de tres funciones: la principal; la encargada de crear la ventana; y la que realiza la inversión del texto, que es la acción deseada al pulsar el botón.

En la función principal inicializamos la "toolkit" necesaria para las funciones de GTK+, llamamos a la función para crear la ventana, ajustamos el tamaño deseado de la misma, activamos los parámetros fijados al caracterizar la ventana, creamos una señal para cerrar la ventana y ejecutamos el bucle principal hasta que demos la orden en el código para salir de él.

En la función para crear la ventana se pueden diferenciar cuatro partes: la declaración

de las variables; la definición de esas variables; la construcción del entorno de la ventana; y la creación de la señal encargada de llamar a la función de invertir el texto al presionar el botón de la ventana.

La función para invertir el texto consta de entrada, implementación de código y salida.

El uso y resultado de ejecutar el programa se muestra en las figuras 16, 17, 18 y 19; y es el siguiente:

Al iniciar el programa:

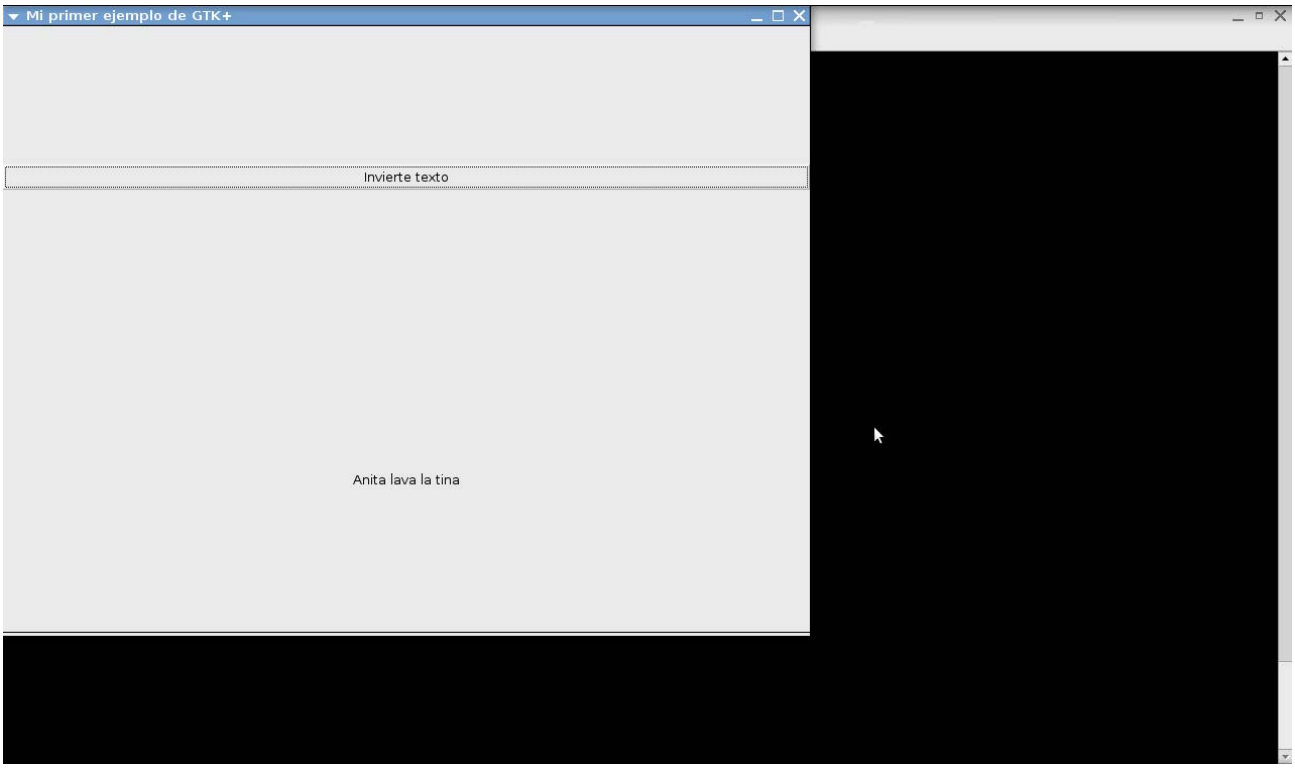

Figura 16 – Inicio de ejecución del “Programa 3”

Al presionar el botón:

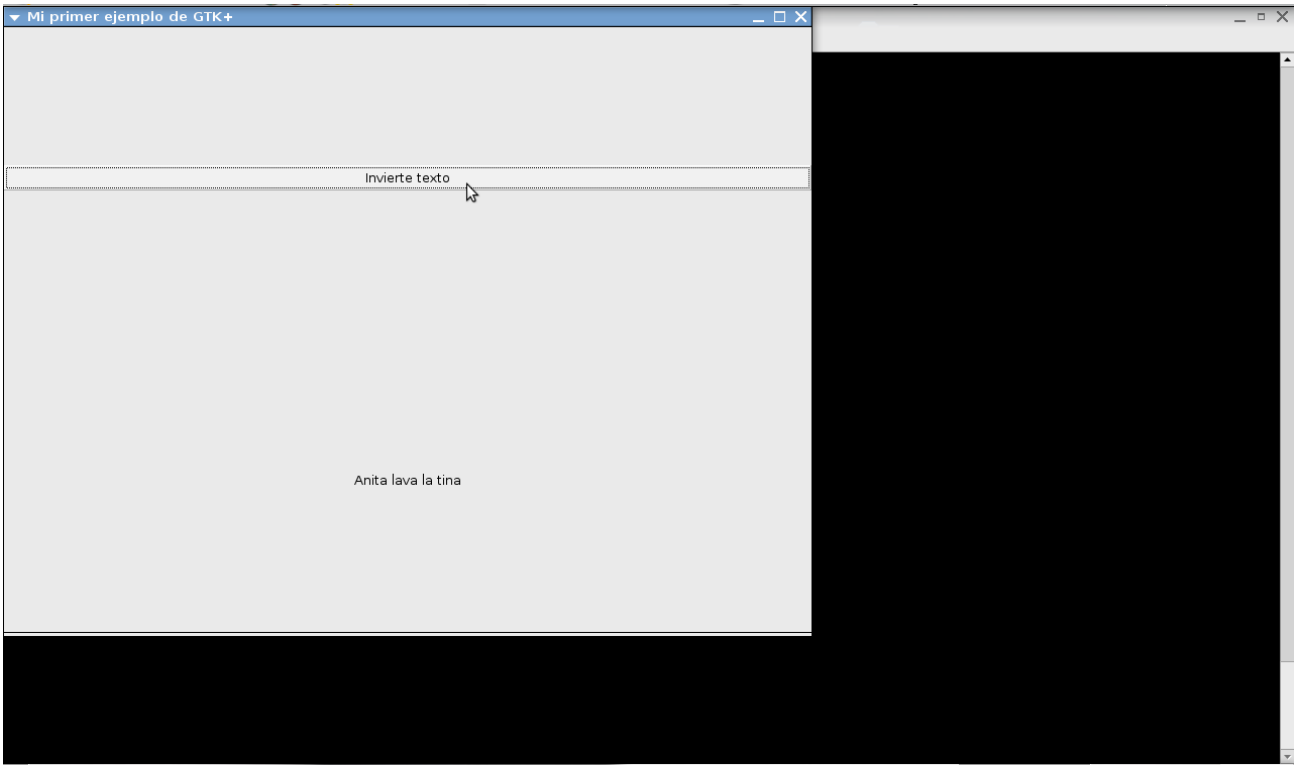

Figura 17 – Selección del botón del "Programa 3"

Se puede observar:

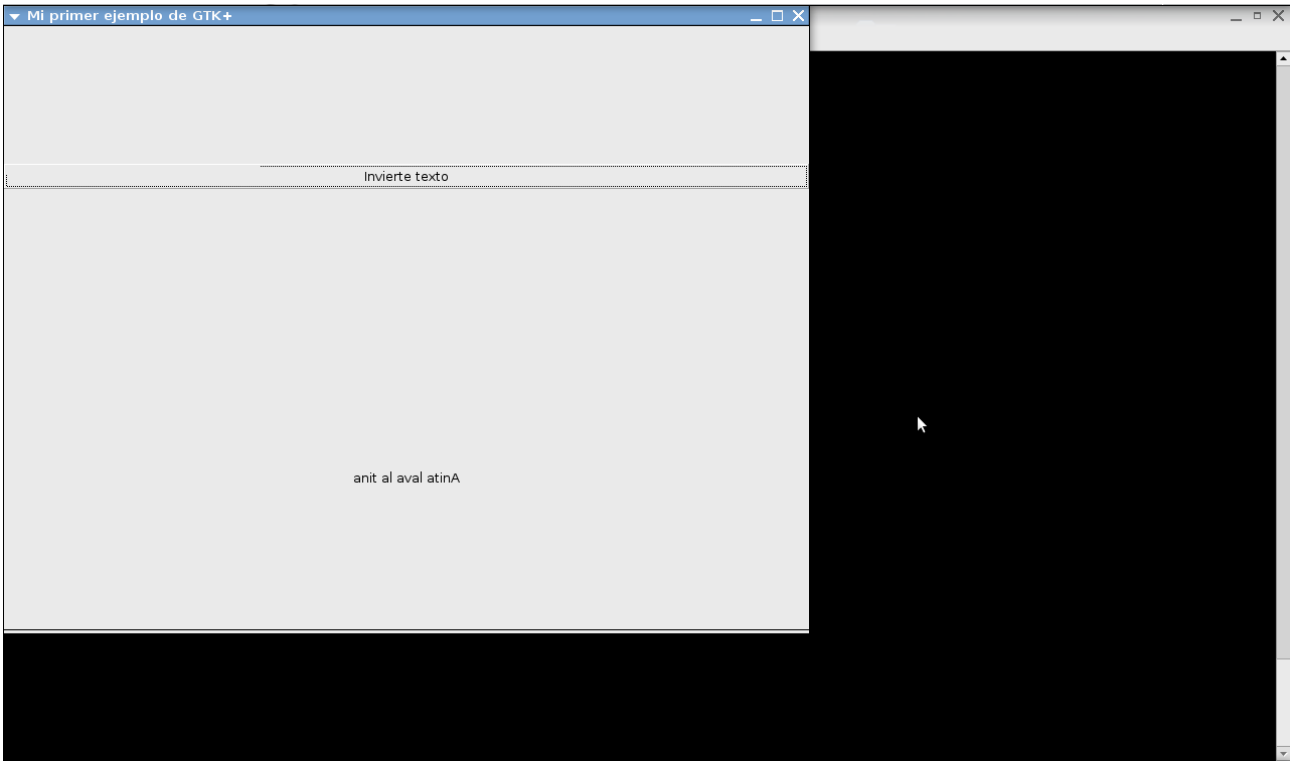

Figura 18 – Resultado de ejecución de selección del "Programa 3"

Para salir del programa presionamos el aspa superior derecha:

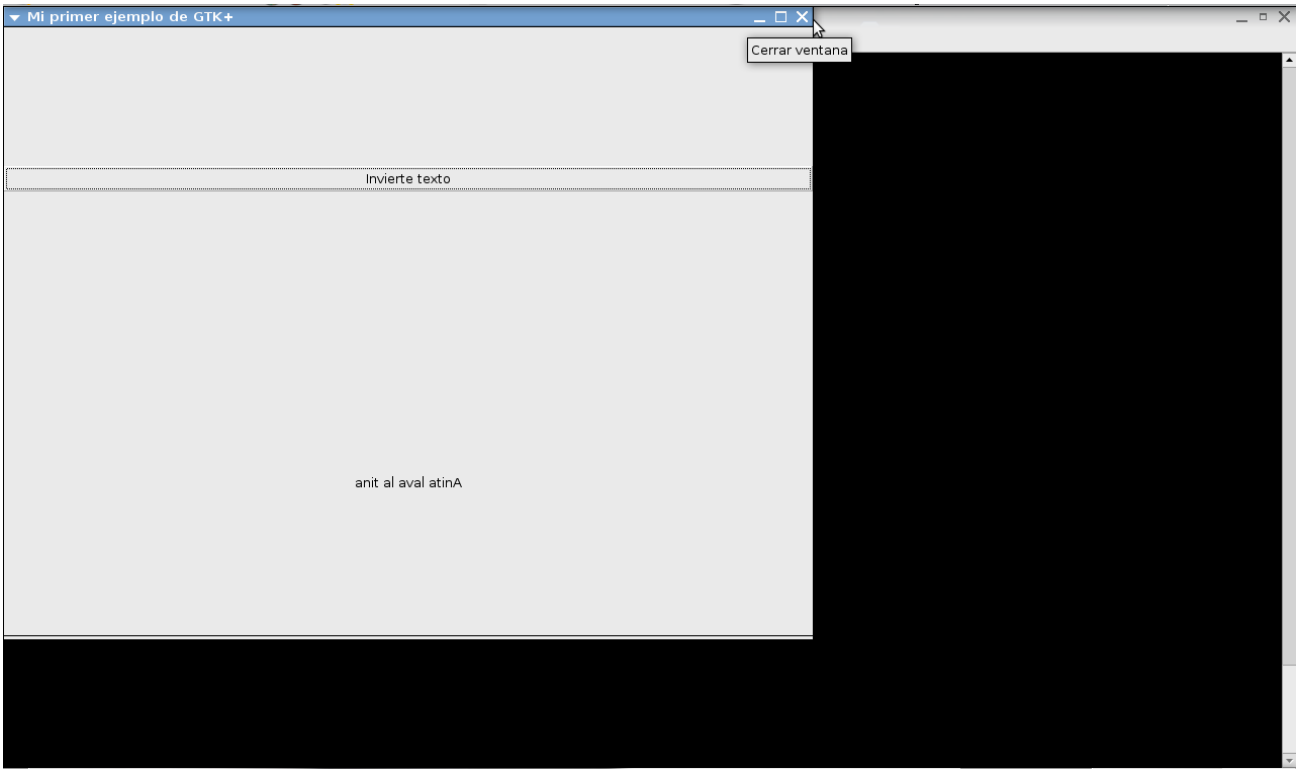

Figura 19– Selección del botón para salir del "Programa 3"

3.2.4.2. Diseño aplicación con el "Programa 2" y GTK+

Una vez realizado el programa del apartado anterior, ya se empezará a diseñar la aplicación final objeto del proyecto. Para empezar, se diseñará un programa que cree una ventana, que muestre la imagen de una cámara de red, que cargue el detector del bote de Coca-Cola y se marquen éstos, en caso de que existan, con un rectángulo. Todo ello implementado con una interfaz gráfica con un botón que ejecuta el inicio de la detección.

Al programa que implementa esta funcionalidad se le ha llamado "Programa 4". Su código se incluye en el Apéndice A.V.

El uso y resultado de la ejecución de este programa se muestra en las figuras 20, 21, 22 y 23; y es el siguiente:

Al inicio:

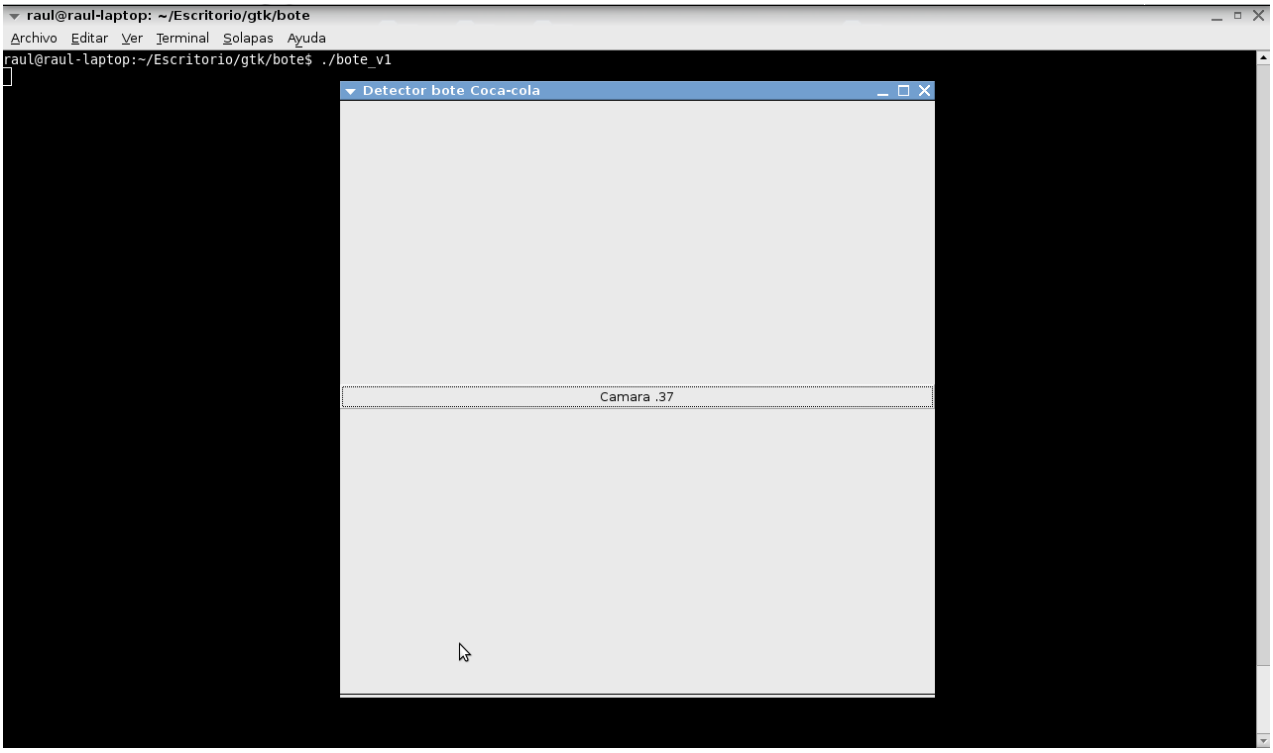

Figura 20 – Inicio de ejecución del "Programa 4"

Al pulsar el botón que activa la visión de la cámara, en este caso sólo hay un botón y por tanto sólo una cámara:

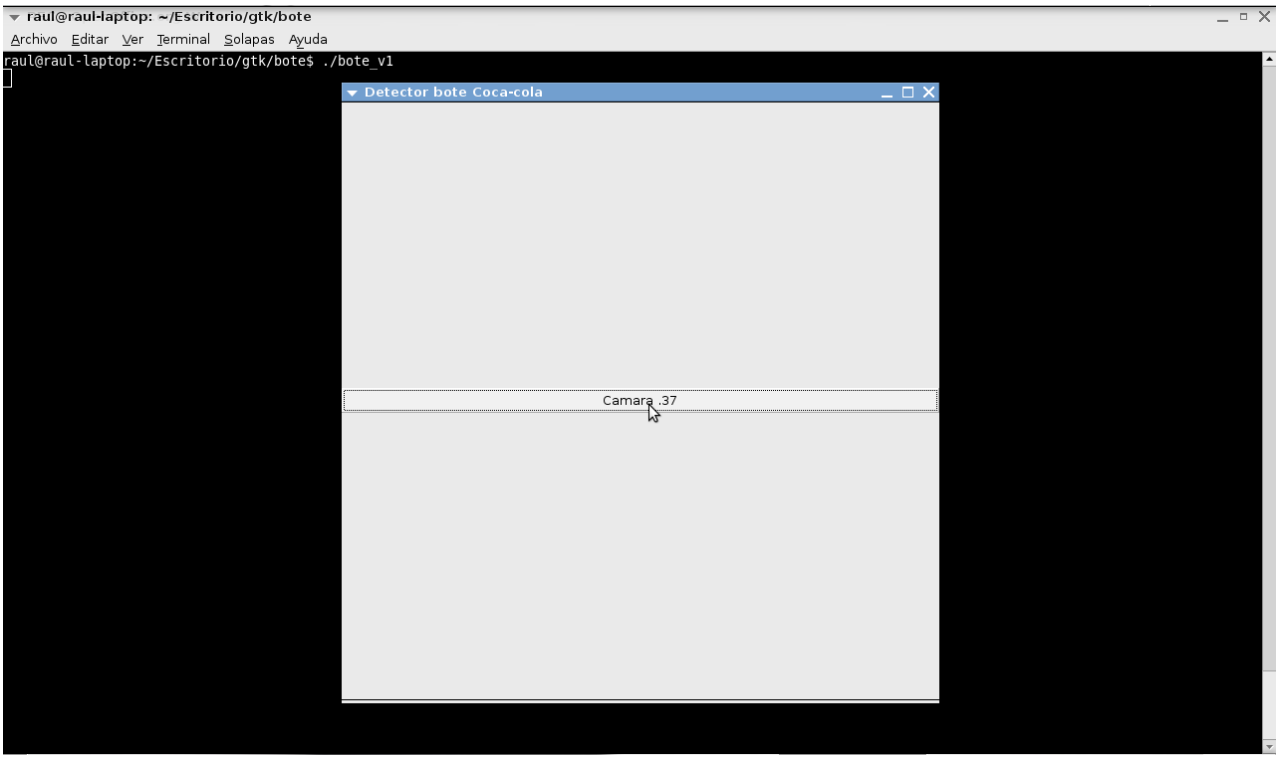

Figura 21 – Selección del botón del "Programa 4"

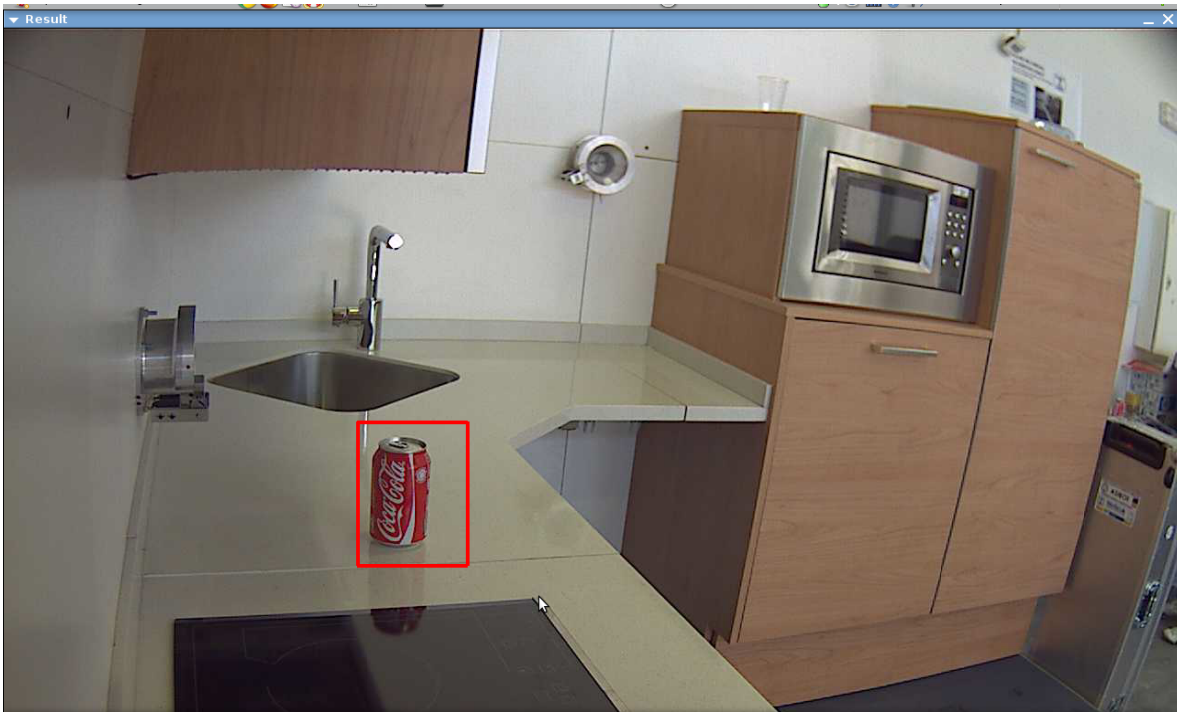

Figura 22 – Resultado de ejecución de selección del “Programa 4”

Para desactivar la imagen de la cámara y el detector pulsaríamos la tecla Esc. Para salir del programa pulsaríamos el aspa de la parte superior derecha de la ventana:

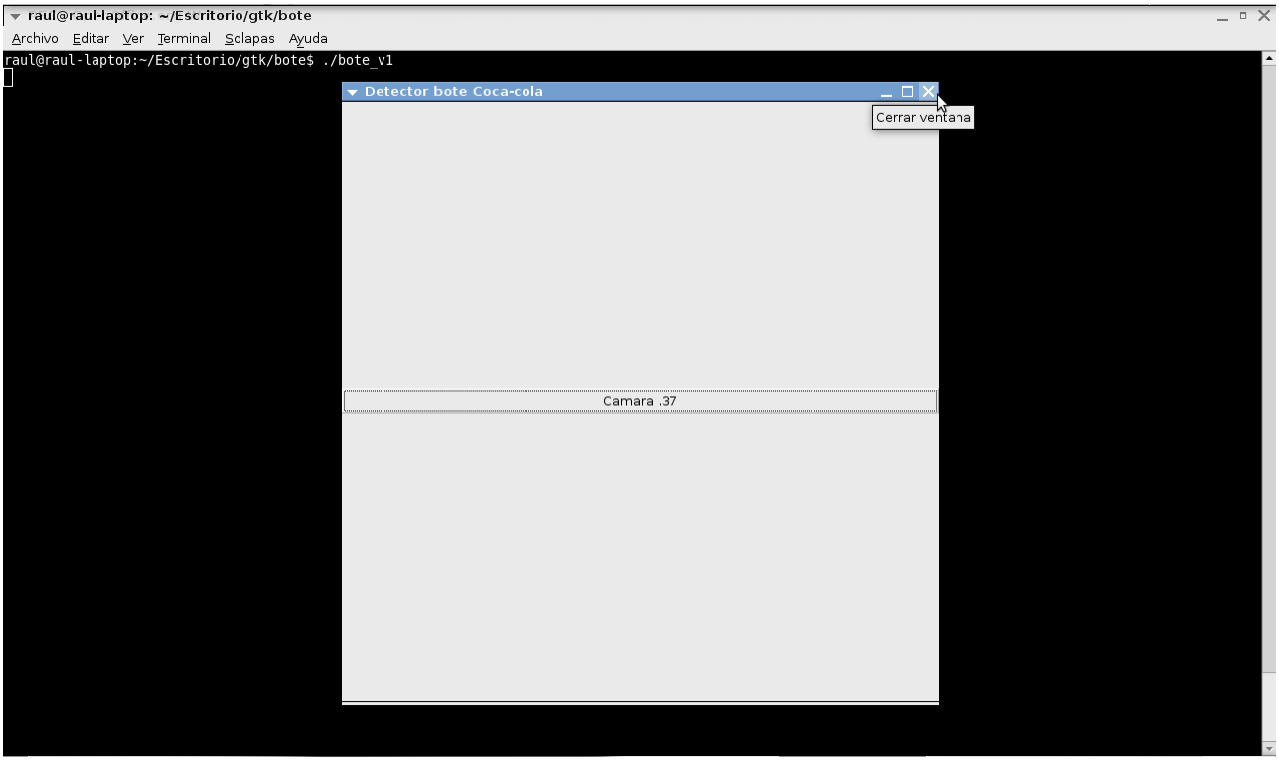

Figura 23 – Selección del botón para salir del “Programa 4”

3.2.4.3. Caracterizar la interfaz gráfica

En el apartado anterior se diseñó el programa base para crear la aplicación final deseada, en éste se va a caracterizar la interfaz gráfica para mejorar su aspecto visual y la funcionalidad del programa aumentando otras opciones dentro del menú del usuario.

El resultado de la ejecución de dicho programa, correspondiente al código incluido en el Apéndice VI, se puede observar en la figura 24.

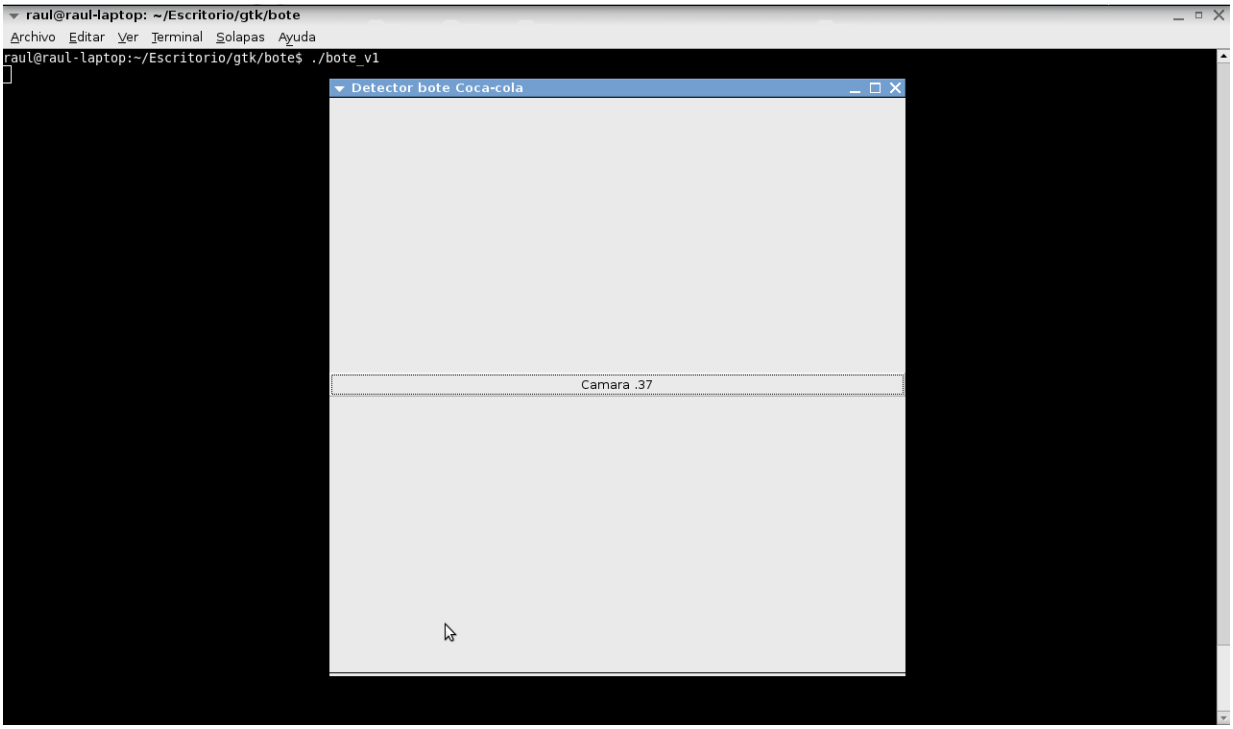

Figura 24 – Imagen de la ejecución del "Programa 4" (programa base)

La primera mejora consiste en cambiarle los colores a la ventana para que tenga mayor impacto visual y añadirle el logotipo de la Universidad Carlos III de Madrid.

El resultado se puede ver en la figura 25.

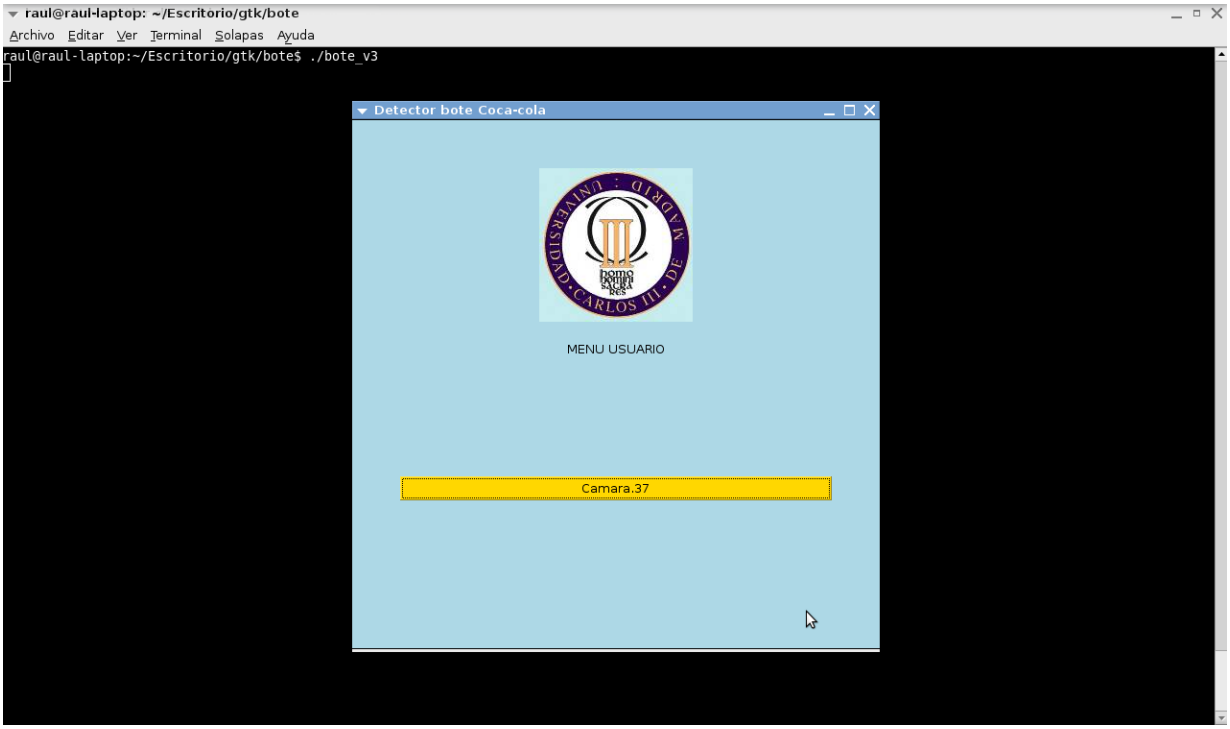

Figura 25 – Ejecución del "Programa 4" con la primera mejora

La siguiente mejora, que afecta tanto al aspecto visual como al funcional, es la creación de un menú de usuario donde poder elegir la opción deseada a ejecutar. En dicho menú se han incluido cuatro botones: tres de ellos para elegir entre una de las tres cámaras de red disponibles; y el cuarto para elegir cargar el detector del bote de Coca-Cola utilizando la imagen captada por la cámara web.

Al ejecutar el programa con esta mejora tendríamos la imagen de la figura 26.

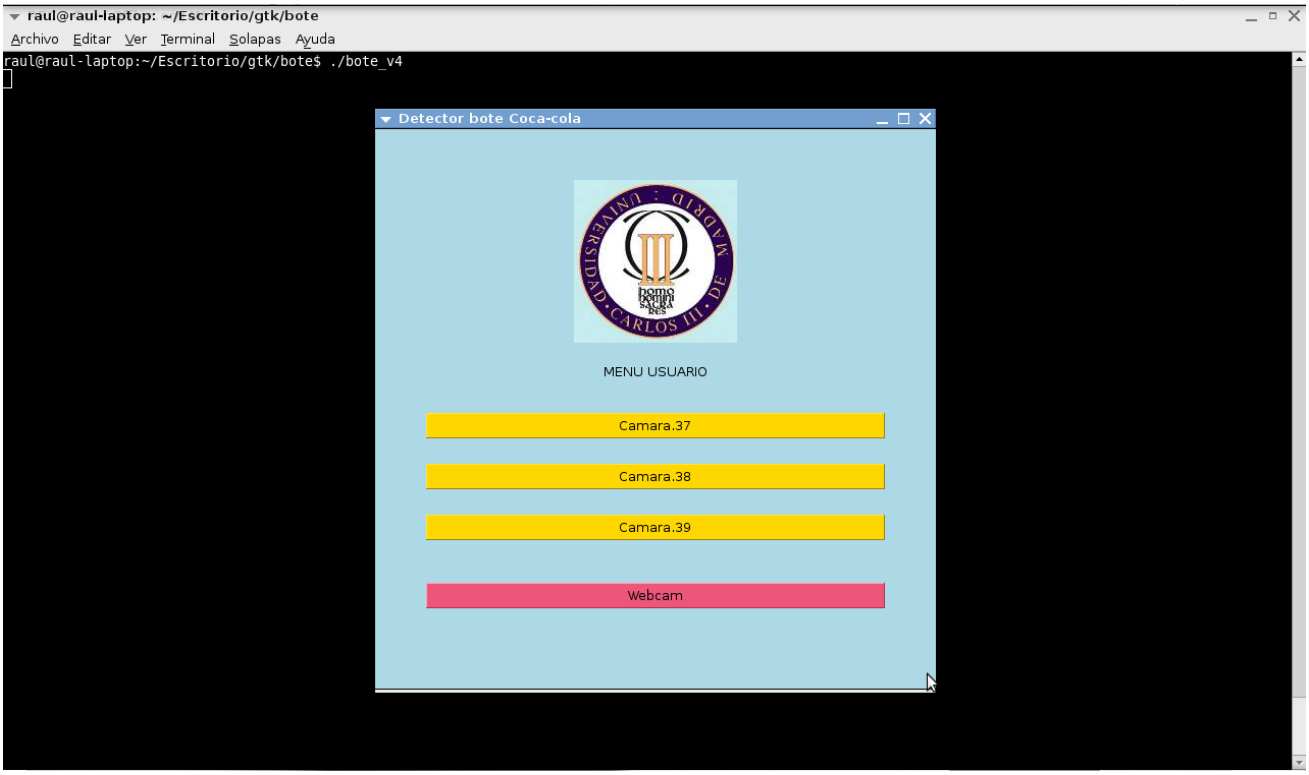

Figura 26 – Ejecución del "Programa 4" con la segunda mejora

La última mejora realizada en la interfaz gráfica es la de añadir otro botón para poder salir del programa sin tener que pinchar en el aspa de la parte superior derecha de la ventana. De este modo queda establecido el menú del usuario completo. Además en cada uno de los botones se ha subrayado la parte que identifica la cámara a utilizar para facilitar su localización.

El resultado es el mostrado en la figura 27.

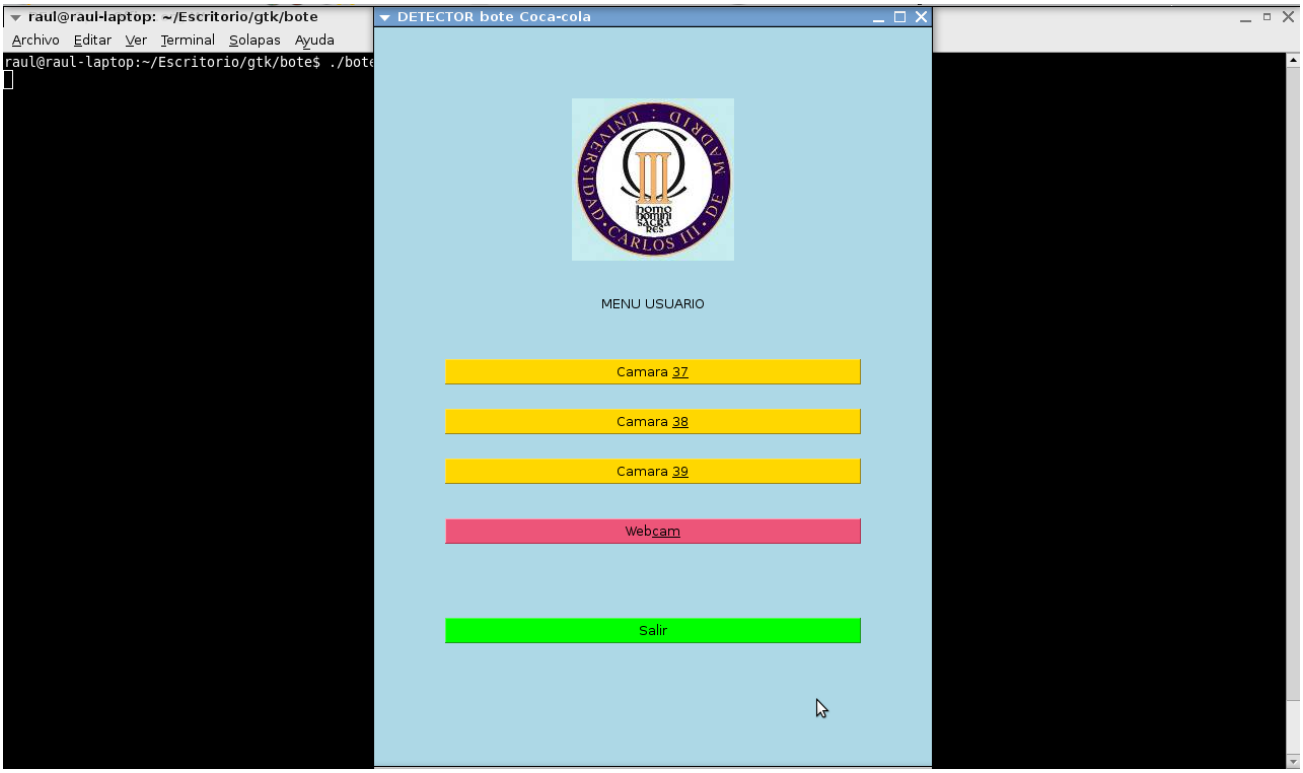

Figura 27 – Ejecución del "Programa 4" con la última mejora

3.2.4.4. Insertar las coordenadas del rectángulo de detección en la pantalla

Para finalizar el diseño del programa requerido se va a realizar una nueva mejora funcional, correspondiente al segundo objetivo establecido para este estudio, que consiste en mostrar las coordenadas espaciales en píxeles de la ubicación del bote de Coca-Cola dentro de la ventana del detector. Estas coordenadas se visualizarán en la parte inferior del cuadro que señala el bote.

Por tanto, la función encargada de la detección quedaría como se muestra en la "Función 1", cuyo código se incluye en el Apéndice A.I.

Para poder visualizar las coordenadas del bote de Coca-Cola en la pantalla ha sido necesario crear una nueva función para convertir las coordenadas de los puntos de la ubicación del bote obtenidas como parámetros enteros a cadenas de caracteres y poder impresionarlas en la pantalla.

En la figura 28 se puede observar el resultado de la ejecución del “Programa 4” con la “Función 1”, para una de las cámaras IP disponibles.

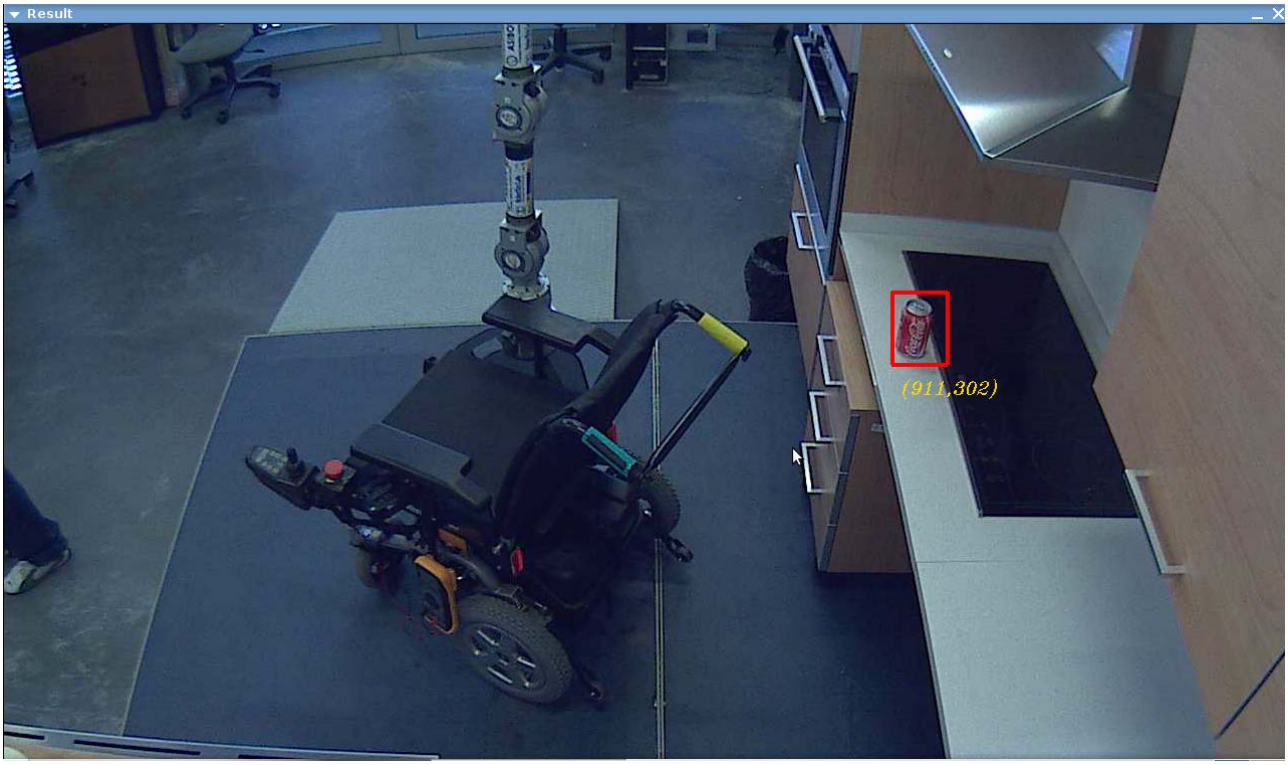

Figura 28 – Ejecución del “Programa 4” con la “Función 1”, para una de las cámaras IP

Una vez realizado, el diseño del programa objeto del presente proyecto vamos a ver un resumen del código en lenguaje C++ del mismo.

El código del programa final completo es el correspondiente al “Programa 5”, cuyo código se incluye en el Apéndice A.VI.

Podemos observar que consta de ocho funciones. La primera es la función principal (main); las tres siguientes corresponden a la activación de la detección de cada una de las tres cámaras de red disponibles; la quinta a la activación de la detección mediante la imagen de la cámara web; la sexta se encarga de salir del programa cuando es activada; la séptima es la función que detecta y ubica mediante sus coordenadas espaciales la presencia del bote de Coca-Cola en la imagen de la cámara seleccionada previamente; y la octava hacer la conversión de parámetro entero a cadena de caracteres necesario para visualizar correctamente las coordenadas del bote en la pantalla.

Capítulo 4

RESULTADO Y FUNCIONAMIENTO

4.1. Análisis de resultados

Una vez ejecutado el programa Haartraining, el clasificador puede ser probado con la utilidad Performance.

La llamada a la aplicación Performance para probar el clasificador del bote de Coca-Cola es la siguiente:

```
opencv-performance -data data/cascade -info positive/info.txt -w 50 -h 65 -ni
```

Donde los argumentos son:

- *data*: Nombre del directorio donde se almacena el clasificador.

- *info*: Precederá al nombre del fichero lista que contendrá todas las imágenes de los objetos que constituyen las muestras positivas.

- *w* y *h*: Tamaño de las muestras (en píxeles). Debe ser el mismo que el usado durante el entrenamiento con la utilidad Haartraining.

- *ni*: Suprime la opción de almacenar las imágenes del archivo de detección.

La salida por pantalla será de la forma:

```
+==================================+======+======+======+
|             File Name            |  Hits  | Missed |  False  |
+==================================+======+======+======+
```

Donde File Name indica el nombre del archivo analizado, Hits muestra el número de objetos correctos encontrados en el archivo, Missed el número de objetos no encontrados, que el programa detecta que existen pero no los encuentra (falsos negativos) y False el número de falsas alarmas, que no existen pero el programa los detecta (falsos positivos). Si el objeto a buscar es encontrado correctamente, la nueva imagen contendrá un rectángulo rojo alrededor del objeto; si devolvió un falso negativo, la imagen será igual a la original y si devolvió un falso positivo, la imagen contendrá un rectángulo rojo situado donde el programa detectó el falso positivo.

Se mostrará una línea por cada muestra positiva y al final se muestra el total del archivo. También podemos observar otros tres parámetros: número de etapas, número de clasificadores débiles y tiempo total. En la figura 29 podemos ver el resumen del resultado obtenido con la aplicación Performance.

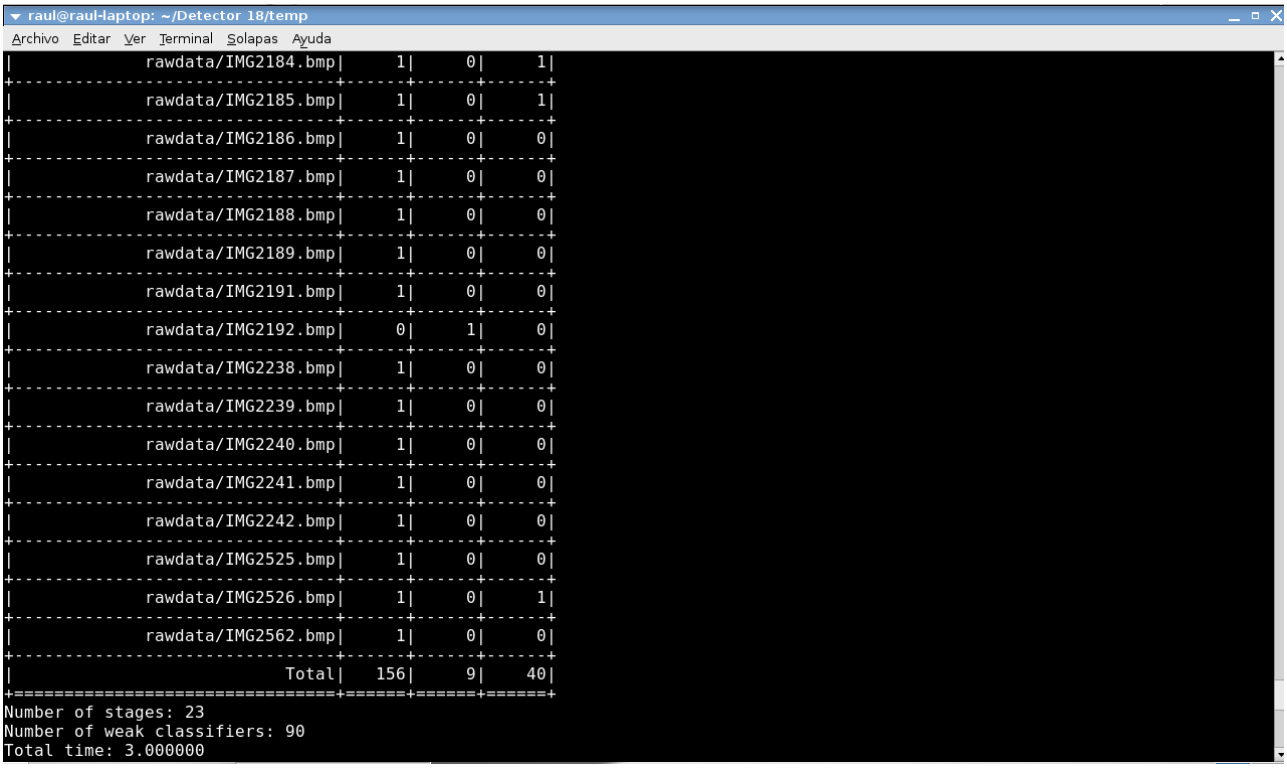

Figura 29 – Resumen del resultado de la aplicación Performance

Por último, se muestra una tabla para construir el diagrama ROC (ver sección 2.7), como se muestra en la figura 30.

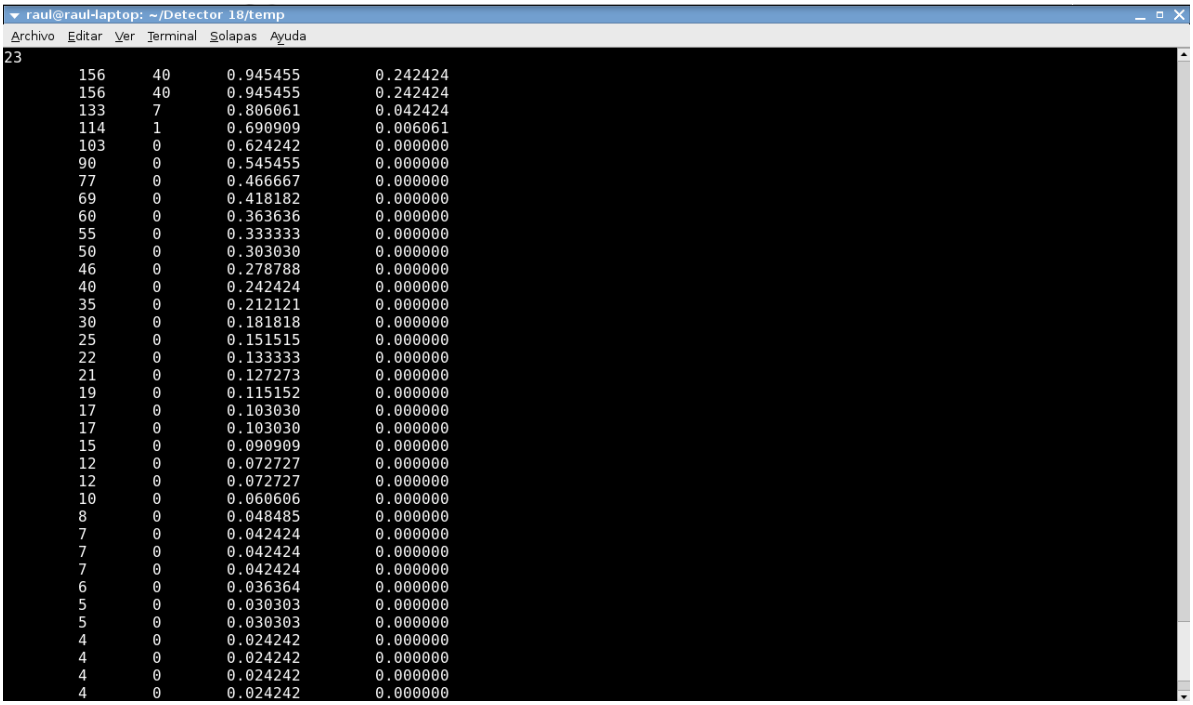

Figura 30 – Tabla obtenida de Performance para construir la tabla ROC

Para dibujar dicho diagrama se puede utilizar el programa Matlab. Para ello se debe usar el siguiente código [27]:

```
ROC=[tabla (todas las líneas)];
plot(ROC(: , 4) ,ROC(: , 3));
```

4.1.1. Detectores

Una vez explicado todo el proceso, desde la adquisición de las muestras hasta la creación del archivo xml del clasificador, y las posteriores pruebas para evaluarlos, se van a comparar los clasificadores creados a lo largo del presente estudio hasta llegar al clasificador final con las mejores características y funcionamiento de detección.

Aunque para la elaboración de este estudio se llevó a cabo la realización de numerosos clasificadores de detección, se van a ver en profundidad solo alguno de ellos, los más representativos. Para cada uno se mostrará el resumen de: muestras positivas, muestras negativas, objetos encontrados, falsos negativos, falsos positivos, número de etapas, tiempo total de prueba y curva ROC. En el CD adjunto a este proyecto se pueden ver las imágenes y características de cada etapa para todos los clasificadores entrenados.

Para analizar la curva ROC del clasificador nos basaremos en dos parámetros: cercanía de la curva con el punto de clasificación perfecta (0,1), es decir, ningún falso negativo y ningún falso positivo; y área entre la curva ROC y la línea de no-discriminación, es decir, la diagonal trazada desde el extremo inferior izquierdo hasta la esquina superior derecha de la gráfica.

- ✓ Detector 1

Muestras positivas--> 1463 (30x30 píxeles)
Muestras negativas--> 2053 (320x240 píxeles)
Objetos encontrados--> 184
Falsos negativos--> 1279
Falsos positivos--> 183
Número de etapas--> 27
Tiempo total--> 129 segundos

Como vemos el nivel de acierto es muy bajo, sólo han sido encontrados 184 objetos y disponemos de 1463 muestras positivas, lo que supone un 12,6% de aciertos, es decir, un 12,6% de posibilidades de que si existe un bote de Coca-Cola en una imagen el clasificador lo encuentre. También observamos que el número de falsos positivos es un poco elevado y el de falsos negativos es muy alto, por tanto, el clasificador no detectaría correctamente un bote de Coca-Cola en la imagen.

La curva ROC, figura 31, no tiene apenas área con respecto a la línea de no-discriminación y dista mucho del punto de clasificación perfecta (0,1).

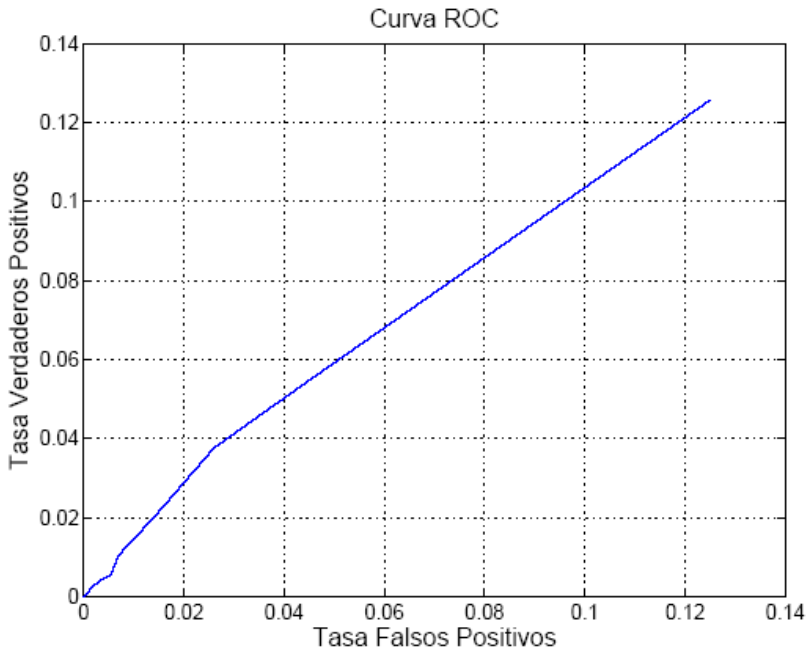

Figura 31 – Curva ROC del detector 1

✓ Detector 3

Muestras positivas--> 2650 (30x30 píxeles)
Muestras negativas--> 8500 (320x240 píxeles)
Objetos encontrados--> 1340
Falsos negativos--> 1310
Falsos positivos--> 1194
Número de etapas--> 29
Tiempo total--> 4674 segundos

Para realizar el entrenamiento de este detector se aumentó considerablemente el número de muestras positivas y negativas, pensando en que cuanto mayor fuese el número de éstas mejor funcionamiento tendría el detector. Como vemos el nivel de acierto ha aumentado llegando a 50,6%, pero aun sigue siendo bajo. También observamos que el número de falsos negativos se ha incrementado ligeramente y el de falsos positivos se ha elevado mucho, por lo que la imagen del detector mostraría una gran cantidad de rectángulos marcando un

bote de Coca-Cola cuando en realidad no hay. El tiempo de prueba se ha visto incrementado sustancialmente debido a que en cada etapa resulta más complicado encontrar y combinar características que clasifiquen bien al conjunto muestral del que se dispone.

La curva ROC, figura 32, tiene mejores características que la anterior pero aun tiene poca área con respecto a la línea de no-discriminación y dista mucho del punto de clasificación perfecta (0,1).

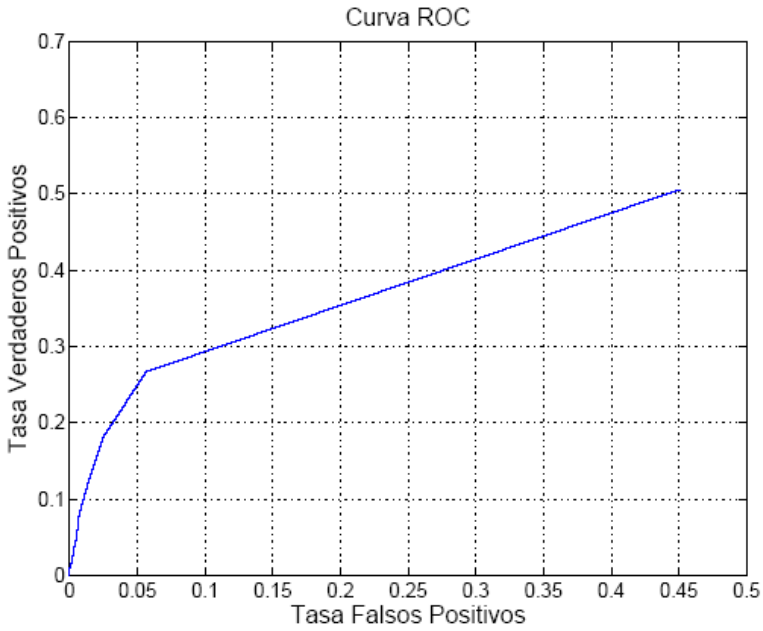

Figura 32 – Curva ROC del detector 3

✓ Detector 8

Muestras positivas--> 200 (25x50 píxeles)
Muestras negativas--> 200 (320x240 píxeles)
Objetos encontrados--> 183
Falsos negativos--> 17
Falsos positivos--> 4961
Número de etapas--> 20
Tiempo total--> 30 segundos

Viendo que el resultado de los anteriores clasificadores no eran satisfactorios se decidió hacer cambios en los parámetros de entrenamiento. En este detector se redujo el número de muestras positivas y negativas y se cambió el tamaño de cuadrado a rectangular, ya que un bote de Coca-Cola no es cuadrado. Realizando estos cambios vemos que el tiempo total es bastante menor a las anteriores pruebas, el nivel de acierto ha aumentado llegando a 91,5%, un valor aceptable para un detector, pero el número de falsos positivos tiene un valor claramente inaceptable. Utilizando este detector se verían en la imagen muchos rectángulos indicando un bote de Coca-Cola donde realmente no lo hay.

En la curva ROC, figura 33, cabe destacar el valor muy alto de la tasa de falsos positivos por lo que las características de este clasificador, aún teniendo una tasa de aciertos aceptable, son muy deficientes.

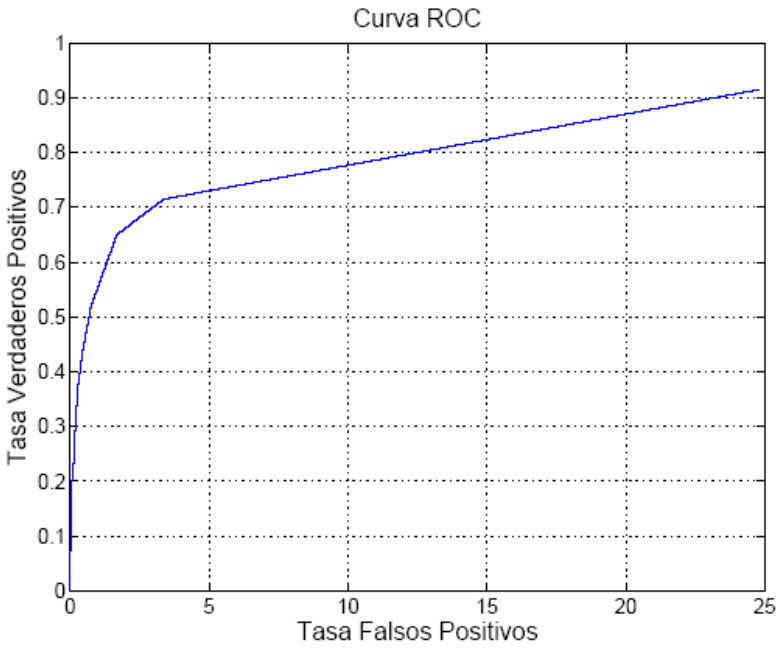

Figura 33 – Curva ROC del detector 8

✓ Detector 10

Muestras positivas--> 10 (25x50 píxeles)
Muestras negativas--> 60 (320x240 píxeles)
Objetos encontrados--> 10
Falsos negativos--> 0
Falsos positivos--> 228
Número de etapas--> 20
Tiempo total--> 1 segundo

Aunque el resultado de este clasificador ya se esperaba, por lo experimentado en el anterior, se hizo otro entrenamiento disminuyendo aún más el número de muestras positivas y negativas. El tiempo total en este caso es menor que en el anterior, tenemos una tasa de acierto del 100%, pero el número de falsos positivos sigue siendo muy alto. Por tanto, este clasificador no reúne las condiciones necesarias para formar parte del detector.

La curva ROC, figura 34, es similar a la anterior, el valor de la tasa de falsos positivos es excesivo para un correcto funcionamiento.

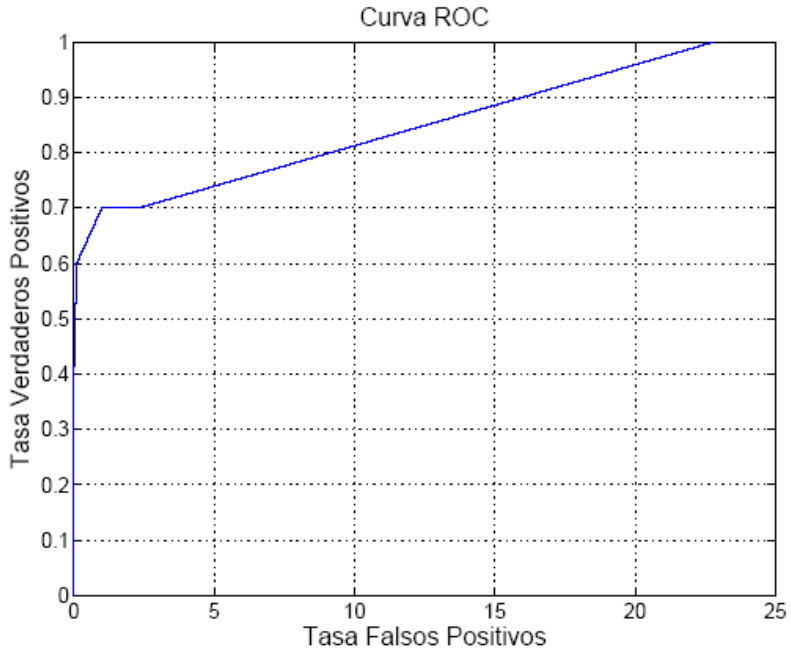

Figura 34 – Curva ROC del detector 10

✓ Detector 18

Muestras positivas--> 165 (50x65 píxeles)

Muestras negativas--> 200 (320x240 píxeles)

Objetos encontrados--> 156

Falsos negativos--> 9

Falsos positivos--> 40

Número de etapas--> 23

Tiempo total--> 5 segundos

Para el entrenamiento de este clasificador se aumentaron tanto el número de muestras positivas como el de negativas. Además se aumentó el tamaño de las muestras creadas para mejorar sus proporciones y facilitar la detección. El resultado es un clasificador con una tasa de acierto del 91,5%, 9 falsos negativos y 40 falsos positivos, por lo que es un clasificador bastante aceptable en cuanto a sus características.

En la curva ROC, figura 35, se puede apreciar que el clasificador es bastante bueno. El eje de la tasa de falsos positivos tiene un valor pequeño, la gráfica está bastante próxima al punto (0,1), punto de clasificación perfecta, y el área entre la curva y la línea de no-discriminación es grande.

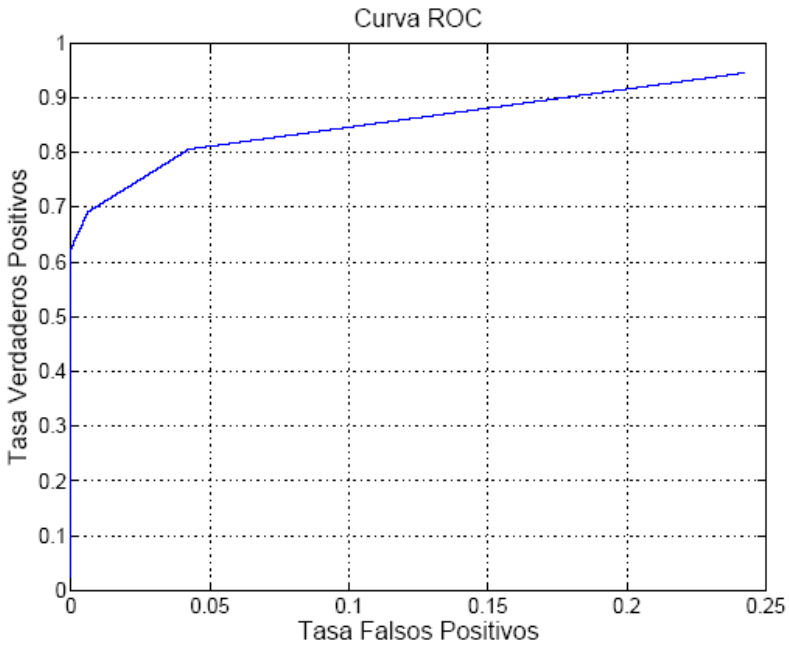

Figura 35 – Curva ROC del detector 18

Se puede afirmar que este clasificador reúne las características suficientes para formar parte de un detector, se va a realizar algún cambio más para intentar mejorarlo y si no es posible este será el clasificador para el detector del bote de Coca-Cola objeto de este estudio.

- ✓ Detector 21

Muestras positivas--> 256 (50x65 píxeles)
Muestras negativas--> 200 (640x512 píxeles)
Objetos encontrados--> 86
Falsos negativos--> 170
Falsos positivos--> 767
Número de etapas--> 29
Tiempo total--> 49 segundos

Hasta ahora para todos los entrenamientos realizados se ha utilizado un tamaño de las imágenes adquiridas para conseguir las muestras positivas y negativas de 320x240 píxeles. Para este clasificador vamos a aumentar el tamaño de las imágenes a 640x512 píxeles, manteniendo el tamaño de las muestras positivas en 50x65 píxeles. También se han incluido más muestras positivas. Vemos que los resultados no son satisfactorios puesto que la tasa de aciertos ha disminuido a 66,4% y el número de falsos positivos a aumentado notablemente.

En la curva ROC, figura 36, cabe destacar la alta tasa de falsos positivos y la baja tasa de verdaderos positivos. Por tanto, esta curva dista mucho del punto de clasificación perfecta (0,1).

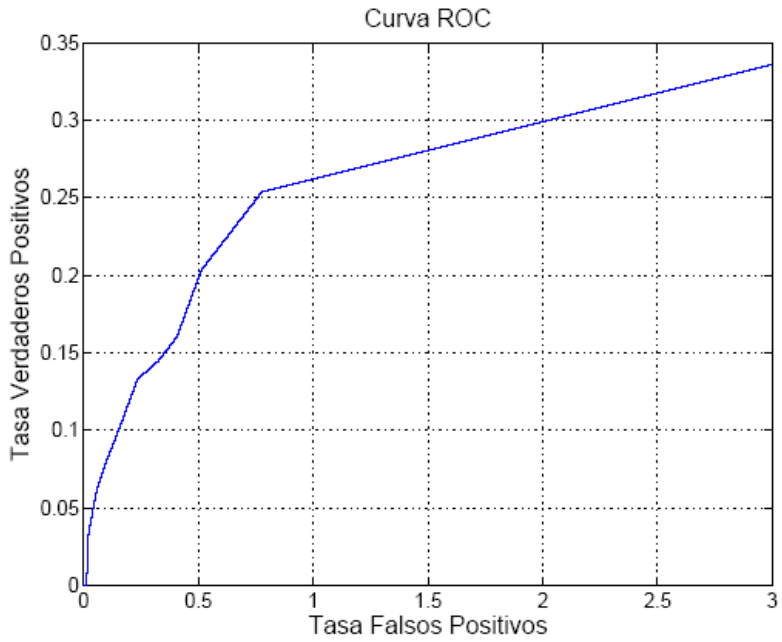

Figura 36 – Curva ROC del detector 21

✓ Detector 25

Muestras positivas--> 136 (50x65 píxeles)
Muestras negativas--> 200 (800x640 píxeles)
Objetos encontrados--> 69
Falsos negativos--> 67
Falsos positivos--> 376
Número de etapas--> 21
Tiempo total--> 24 segundos

En este último entrenamiento explicado, se aumentan aun más el tamaño de las imágenes utilizadas para crear las muestras a 800x640 píxeles y disminuimos el número de muestras positivas. El resultado no difiere mucho del anterior clasificador ya que se alcanza una tasa de aciertos del 50,7% y el numero de falsos positivos se ha disminuido muy ligeramente.

La curva ROC, figura 37, no presenta tampoco grandes cambios con respecto al anterior, la tasa de falsos positivos es muy grande y la curva se aleja mucho del punto de clasificación perfecta (0,1).

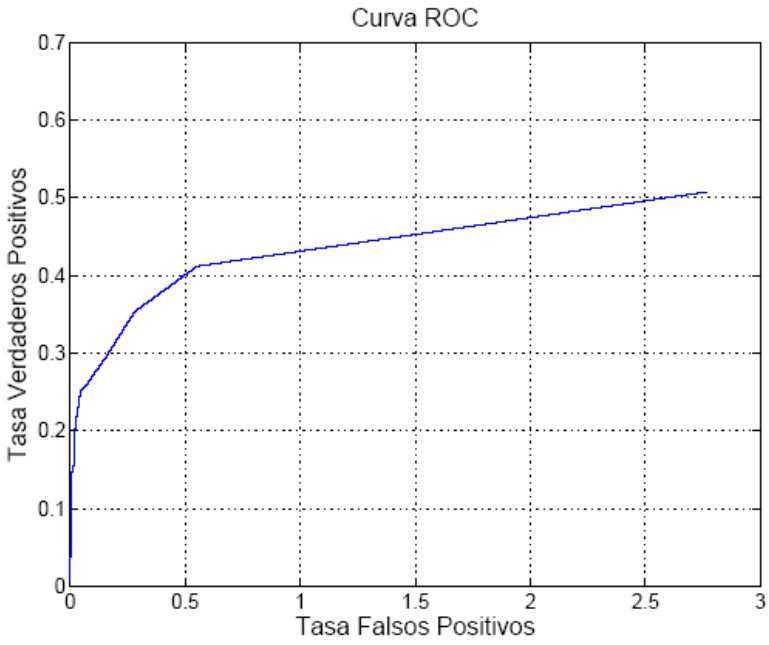

Figura 37 – Curva ROC del detector 25

4.2. Conclusiones de resultados

Después de realizar el entrenamiento a diferentes clasificadores, cambiando varios parámetros en cada uno de ellos y analizando los resultados, llegamos a la conclusión de que el mejor clasificador para formar parte del detector del bote de Coca-Cola es el Detector 18. Para ello se utilizaron 165 muestras positivas y 200 negativas.

Una vez elegido el clasificador a utilizar y creado el archivo xml del entrenamiento se va a pasar a diseñar la aplicación informática para crear el detector del bote de Coca-Cola. Para llegar a la aplicación final se irá diseñando paso a paso, primero se mostrará la imagen de la cámara de red y se cargará el detector y después se implementará la interfaz gráfica hasta llegar al resultado final requerido en este proyecto.

El funcionamiento y resultado de ejecución del "Programa 5" se puede observar a lo largo de la secuencia de imágenes entre la figuras 38 y 47. Al ejecutar el programa aparece la pantalla con el menú del usuario para elegir la opción deseada:

Figura 38 – Ejecución del "Programa 5"

Si presionamos uno de los tres botones primeros, marcados con amarillo, activamos la detección del bote de Coca-Cola en la imagen de la cámara de red seleccionada:

Figura 39 – Selección para ejecutar la primera cámara en el "Programa 5"

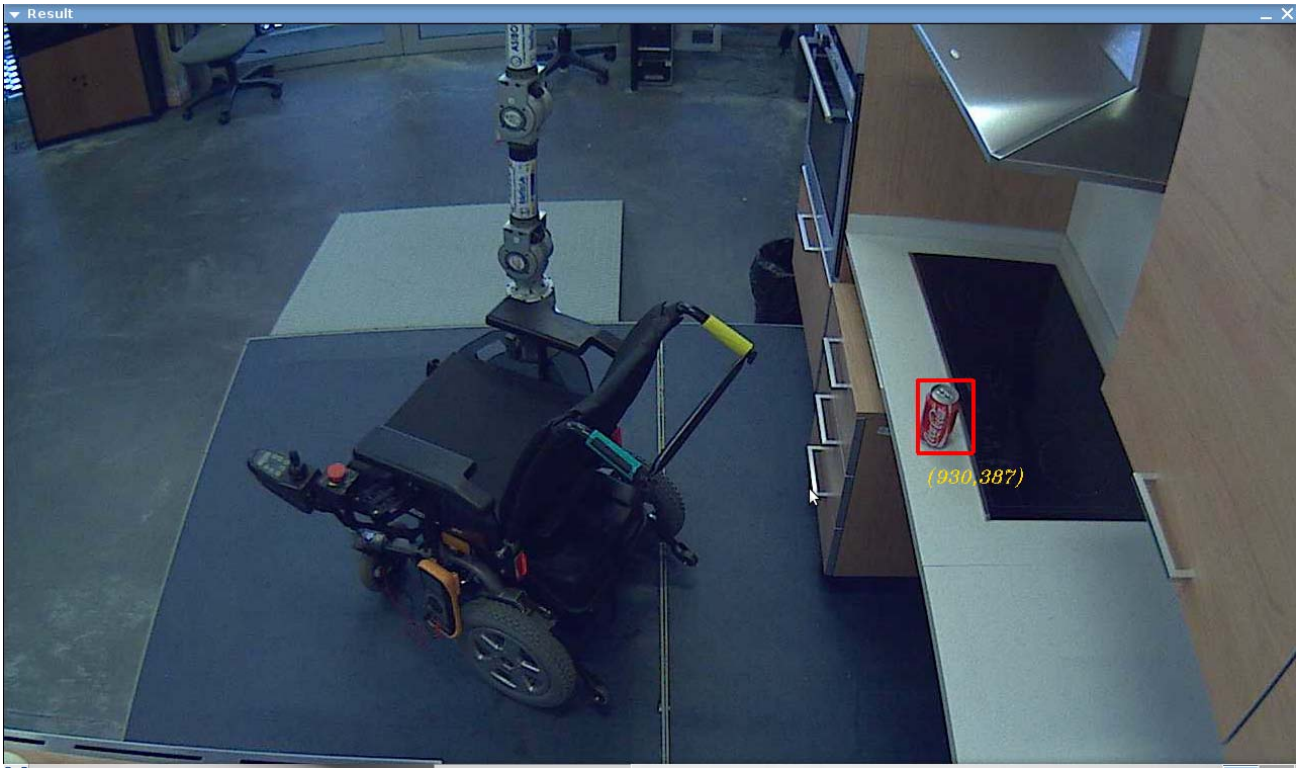

Figura 40 – Resultado de ejecución de la primera cámara en el "Programa 5"

Figura 41 – Selección para ejecutar la segunda cámara en el "Programa 5"

Figura 42 – Resultado de ejecución de la segunda cámara en el "Programa 5"

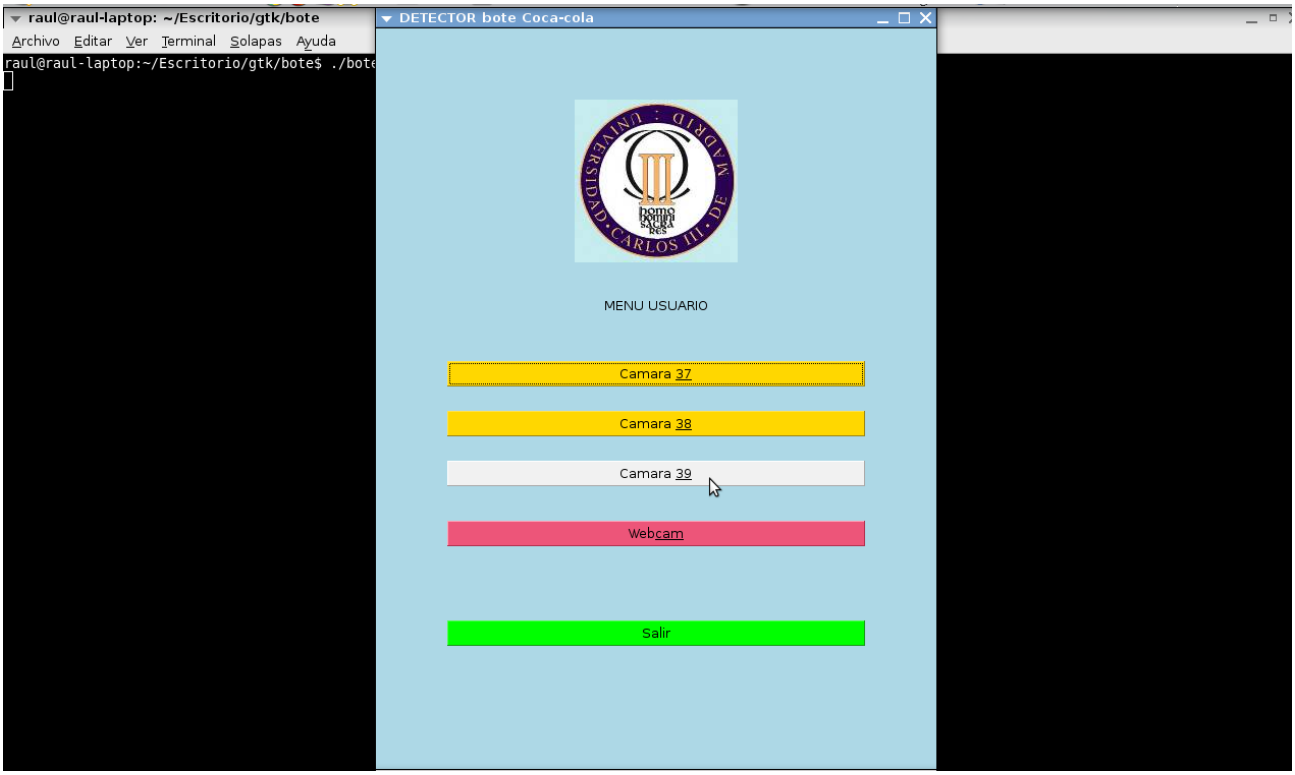

Figura 43 – Selección para ejecutar la tercera cámara en el "Programa 5"

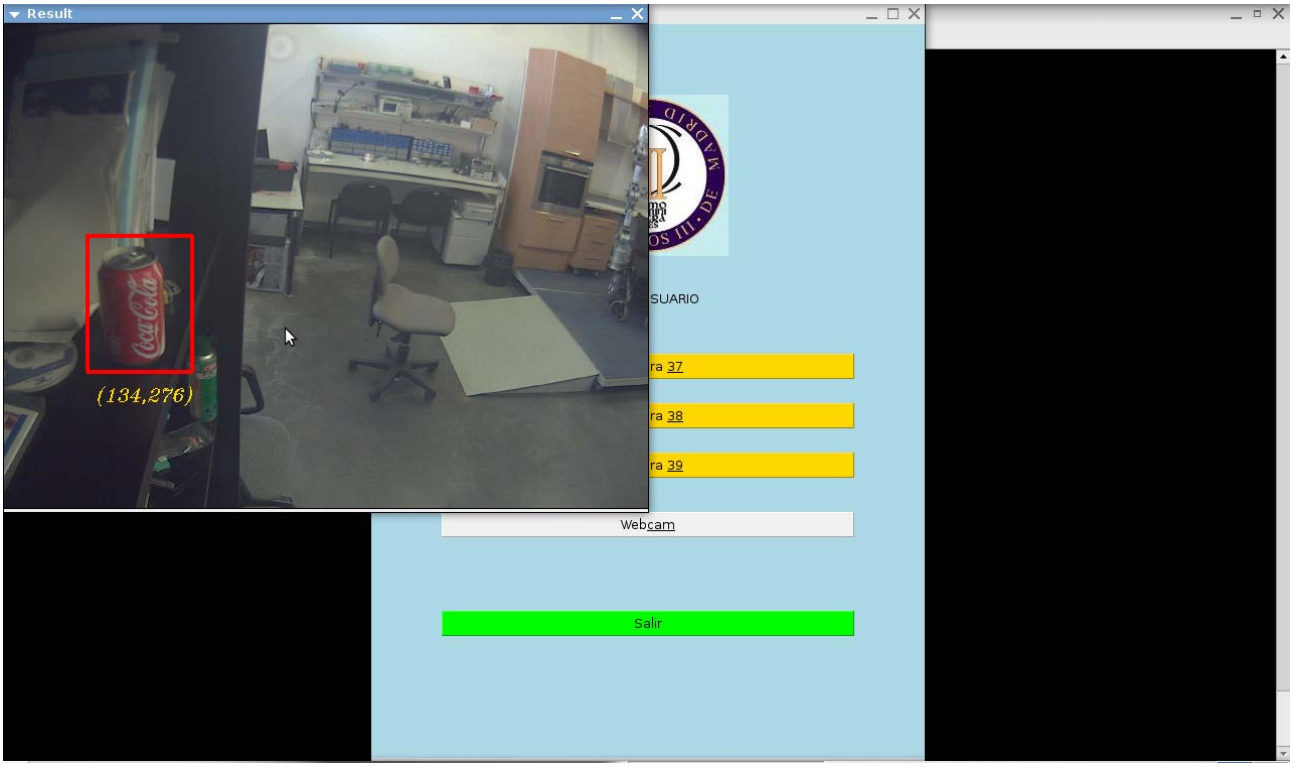

Figura 44 – Resultado de ejecución de la tercera cámara en el "Programa 5"

Si presionamos el cuarto botón, marcado con magenta, activamos la detección del bote de Coca-Cola en la imagen de la cámara web.

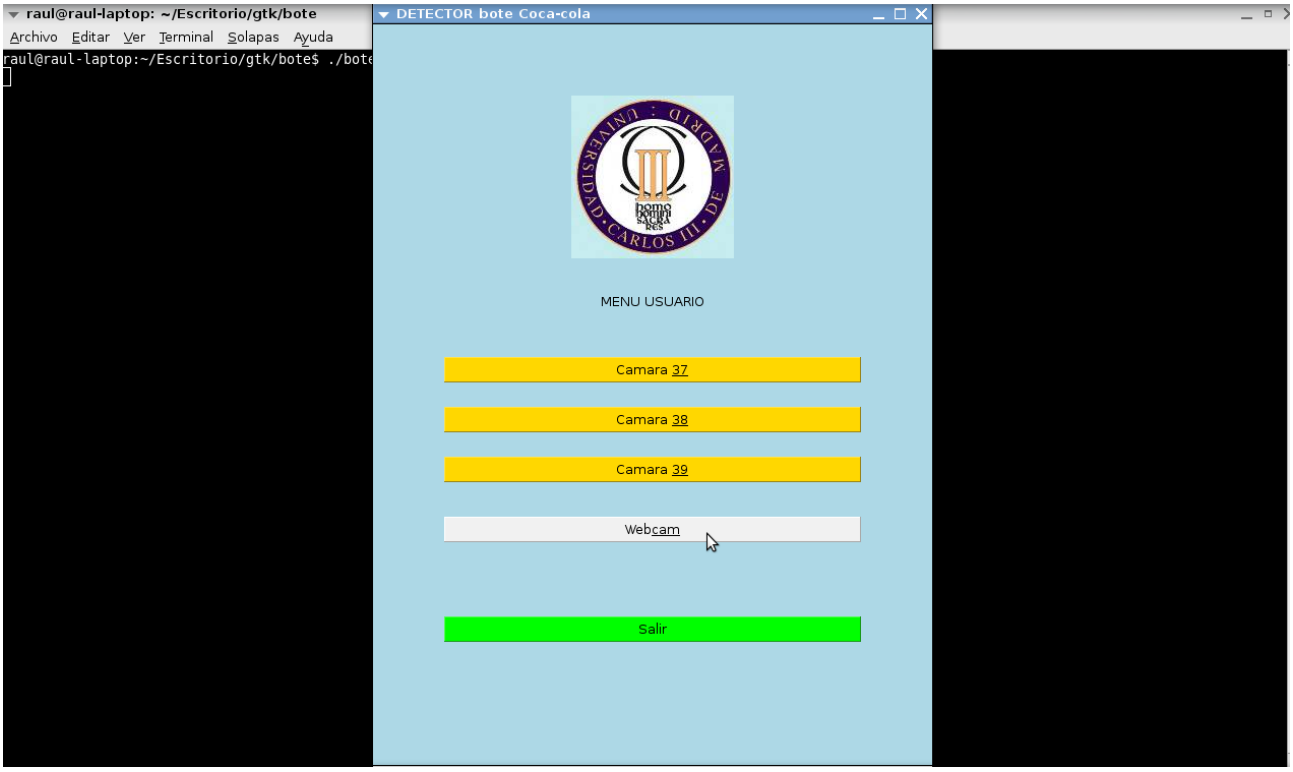

Figura 45 – Selección para ejecutar la cámara web en el "Programa 5"

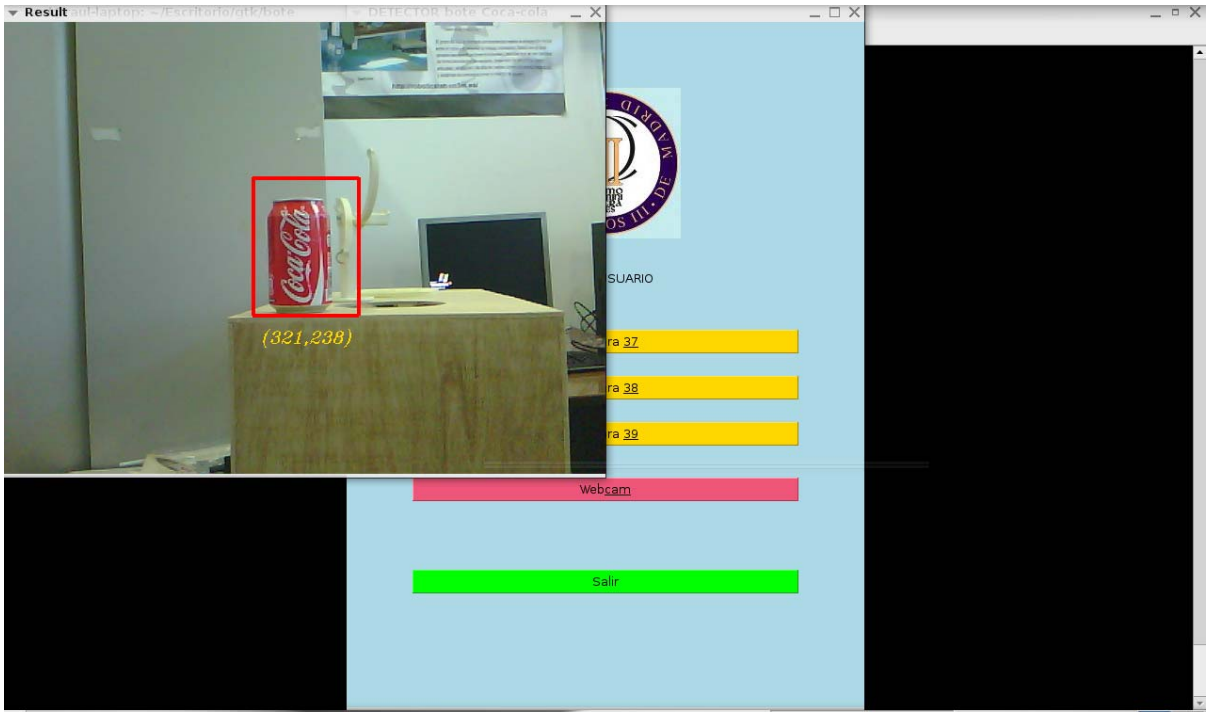

Figura 46 – Resultado de ejecución de la cámara web en el "Programa 5"

Y si presionamos el último botón, marcado con verde, salimos del programa.

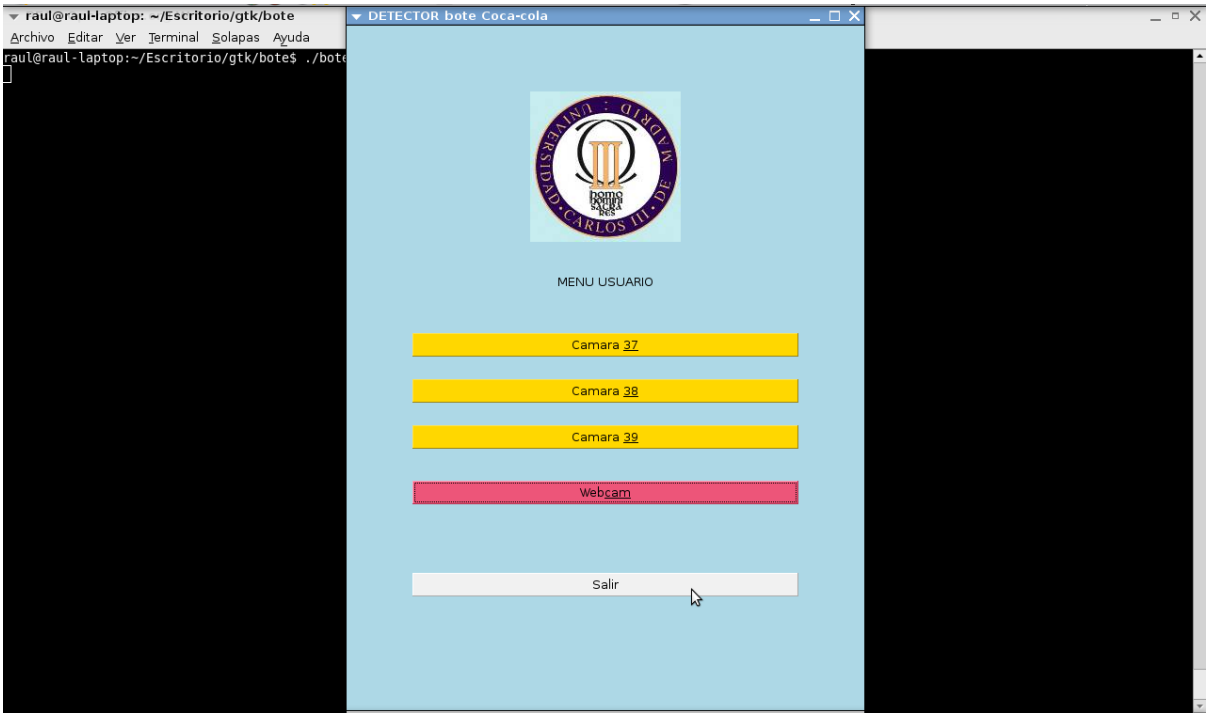

Figura 47 – Selección del botón para salir del "Programa 5"

Por último, comentar que si durante la ejecución del detector, independientemente de la cámara seleccionada, pulsamos la tecla Esc volvemos al menú del usuario.

Capítulo 5

CONCLUSIONES Y POSIBLES MEJORAS

Para finalizar este proyecto se van a comentar las conclusiones obtenidas tras todo el estudio realizado, tanto de las aplicaciones utilizadas como del propio desarrollo del mismo. Primero, se presentará un pequeño análisis crítico; segundo, se comentarán las conclusiones de todo el desarrollo del proyecto, desde el planteamiento de los objetivos hasta la obtención de los resultados satisfactorios cumpliéndolos; y por último, se propondrán unas posibles mejoras para estudiarlas en trabajos futuros que pueden mejorar, o ampliar los objetivos planteados en este trabajo.

5.1. Análisis crítico

La herramienta fundamental y en la que se basa este proyecto, la creación de un sistema detector de objetos, es el entrenamiento necesario para construir el clasificador, que será el encargado de decidir si en una imagen está el objeto o no. En este estudio, como ya se ha explicado durante el mismo, se ha utilizado el Haartraining [4] para construir dicho clasificador.

A continuación se van a comentar los principales inconvenientes y ventajas encontradas durante la utilización del Haartraining.

- ➔ Inconvenientes:
  - o Hay que seguir unas etapas de funcionamiento estricta y meticulosamente para llegar a un buen resultado.
  - o Hay que capturar varias imágenes del objeto a detectar, con pequeñas variaciones de perspectiva y en diferentes condiciones ambientales., lo

que supone trasladar el objeto ha diferentes lugares con entornos diferentes.

- A la hora de marcar manualmente donde se encuentra el objeto a detectar en las imágenes (en este proyecto se ha realizado mediante la aplicación denominada Objectmarker [24]), hay que hacerlo una a una y todas ellas, lo que supone un trabajo bastante laborioso.

- Construir el clasificador supone realizar por parte del computador muchos cálculos y procesamiento de datos y eso conlleva dejarlo trabajar sin pausa durante bastante tiempo. Para crear un clasificador que forme parte de un detector de objetos aceptable, un PC doméstico puede tardar varios días o incluso semanas en procesar toda la información.

➔ Las principales y más importantes ventajas encontradas con las siguientes:

- Siguiendo correctamente los pasos necesarios se puede construir un detector de objetos con buena fiabilidad.

- Se pueden diseñar detectores para diferentes objetos, simplemente hay que realizar el entrenamiento con las muestras correspondientes a ese objeto.

- Se puede integrar el detector en sistemas reales, como es el caso de este proyecto, por lo que aumenta su utilidad.

- Una vez realizada la aplicación final es de muy fácil utilización. Esto permite adaptarse a las diferentes condiciones del usuario.

5.2. Conclusión

El proyecto tenía principalmente dos objetivos, el primero diseñar una aplicación informática ejecutable bajo entorno Linux que mostrase la imagen de una cámara de red (cámara IP), que detectara la presencia de un bote de Coca-Cola y lo señalizara en la ventana creada reflejando información de sus coordenadas en la pantalla. Y el segundo, implementar esa aplicación en una interfaz gráfica, facilitando su utilización al usuario y permitiendo la elección de varias cámaras y opciones.

Durante todo el desarrollo del proyecto se ha estado trabajando con el método del Boosting [17] y con el programa que lo lleva a cabo para conseguir el objetivo de detección, el Haartraining.

Se ha podido comprobar el efecto del Boosting; cómo, a partir de reglas tan sencillas se puede llegar a construir una estructura jerárquica de decisión capaz de discernir, con buen criterio, el objeto buscado. El método requiere como entrada principal el conjunto muestral del que aprende las características del objeto que se desea reconocer. Éste se ha de componer como se ha visto, de dos subconjuntos; el positivo que contiene ejemplos del objeto y el negativo, con imágenes que no lo contengan. Tanto la calidad como la variedad de ambos subconjuntos son muy importantes para poder generar un clasificador que sea aceptable para formar parte del detector.

Hay cuatro parámetros importantes en el comando del entrenador que son los que se han modificado para construir los distintos clasificadores: el tamaño de las imágenes reales capturadas, el número de muestras positivas, el número de muestras negativas y el tamaño de las muestras creadas. En el CD adjunto a este proyecto se pueden observar las características e imágenes de los distintos detectores creados desde el inicio del estudio hasta llegar a la elección del mejor. En el capítulo “Resultado y funcionamiento” del documento se comentan los resultados y la evolución de los detectores más representativos del estudio.

Para obtener los valores óptimos de los parámetros del entrenamiento es necesario realizar distintos entrenamientos con diferentes valores y analizar los resultados.

En los primeros clasificadores se utilizaron imágenes de muy variadas características con el objetivo de producir un efecto de gran generalización en el clasificador final, con el pensamiento de que al disponer de mayor variedad el detector final sería más potente, con mayor capacidad para detectar el objeto bajo condiciones más diferentes. Sin embargo, esto produce precisamente el efecto contrario, al disponer de demasiada variedad el algoritmo no es capaz de extraer las propiedades importantes que caracterizan al objeto. Por tanto, el número de muestras positivas (en este caso 165) debe ser el necesario para contener imágenes del objeto a detectar variando un poco la perspectiva y con diversas condiciones de fondo. Cuanto más se varíen las condiciones ambientales mejor será el detector. El número de muestras negativas (en este caso 200) puede ser aproximadamente igual al de positivas o ligeramente superior para que el entrenador tenga mayor capacidad de decisión.

El tamaño de las imágenes reales del objeto tomadas con la cámara (para este proyecto 320 x 240 píxeles) dependen del tamaño del objeto que se quiere detectar. Si el tamaño del objeto es muy grande el tamaño debe ser superior a si el objeto es pequeño. Al igual ocurre con el tamaño de las muestras creadas (50x65 en este estudio), debe de tener un valor proporcional al tamaño del objeto en realidad. Si tomamos muestras demasiado grandes el entrenador no puede encontrar las características que definen al objeto al estar éste muy distorsionado. Y por el contrario, si las creamos muy pequeñas el entrenador no puede obtener características relevantes del objeto. Para obtener el valor correcto para ambos tamaños es necesario realizar varios entrenamientos y analizar los resultados obtenidos de cada uno de ellos comparándolos con el resto y elegir el óptimo.

Este proyecto ha supuesto retos ya que el programa de entrenamiento de los clasificadores fue más complejo de tratar de lo que podía parecer al principio e hizo que costara cierto esfuerzo sacarlo adelante y empezar a hacer pruebas más frecuentes.

En cuanto al desarrollo del segundo objetivo del proyecto también hubo que realizar un gran trabajo de investigación por la poca experiencia con las interfaces gráficas.

Al final, con mucho esfuerzo y dedicación se ha conseguido alcanzar los objetivos propuestos en este proyecto, consiguiendo una aplicación funcional y fácil de utilizar.

5.3. Trabajos futuros

Los objetivos establecidos en un principio se han cumplido, aunque se podrían realizar algunas mejoras para perfeccionar aun más el resultado. Se van a proponer diversas formas para mejorar el funcionamiento de la aplicación. A continuación se listan los posibles trabajos futuros a investigar:

→ Aumentar la estabilidad del sistema, de forma que el funcionamiento de la aplicación sea más efectivo.

→Mejorar la robustez del sistema clasificador-detector, de manera que se aumente la probabilidad de detección positiva.

→ Realizar el mismo estudio para otros objetos, como por ejemplo: un plato, un vaso, etc.

→Diseñar una interfaz gráfica integrando la imagen final y el menú de funcionamiento del usuario en una misma ventana.

→Integrar este sistema detector con el resto de la arquitectura software del robot ASIBOT, que en la actualidad está basada en YARP [28].

REFERENCIAS Y BIBLIOGRAFÍA

[1] Alberto Jardón; *Metodología de robots asistenciales. Aplicación al robot portátil ASIBOT*; Tesis doctoral 2006, Universidad Carlos III de Madrid.

[2] OpenCV, definición Wikipedia:
http://es.wikipedia.org/wiki/OpenCV
(Última visita Febrero 2011)

[3] GTK+, definición Wikipedia:
http://es.wikipedia.org/wiki/GTK%2B
(última visita Febrero 2011))

[4] Tutorial: OpenCV Haartraining (Rapid object detection with a cascade of boosted classifiers based on haar-like features):
http://note.sonots.com/SciSoftware/haartraining.html
(Última visita Febrero 2011)

[5] Raúl Palencia; *Migración de la plataforma a bordo del robot asistencial ASIBOT*; Proyecto final de carrera 2009, Universidad Carlos III de Madrid.

[6] Eftring H., Boschian K.; *Technical results from manus user trials*; Proc. 6th International Conference on Rehabilitation Robotics, 136-141; 1999

[7] Z. Bien, M. Chung, P. Chang, D. Kwon, D. Kim, J. Han, J. Kim, D. Kim, H. Park, S. Kang, K. Lee, S. Lim.; *Integration of a Rehabilitation Robotic System (KARES II) with Human-Friendly Man-Machine Interaction Units*; Autonomous Robots 16, 165–191; 2004.

[8] ¿Qué es una cámara IP?:
Axis Communications; *¿Qué es una Cámara de Red?*; 2002

[9] Cámara IP Axis 207:
http://www.camara-ip.es/camara-ip-207.htm
(Última visita Febrero 2011)

[10] Arturo de la Escalera; *Visión por computador: fundamentos y métodos*; Prentice Hall, 2001

[11] Explicación teorema de Bayes y ejemplos:
http://www.mitecnologico.com/Main/ReglaDeBayes
(Última visita Febrero 2011)

[12] Metodo K-nn, definición Wikipedia:
http://es.wikipedia.org/wiki/Knn
(Última visita Abril 2011)

[13] Página oficial de Willow Garage:
http://www.willowgarage.com/
(Última visita Febrero 2011)

[14] V.M. Arévalo, J. González, G. Ambrosio; *La librería de visión artificial OpenCV – Aplicación a la docencia e investigación*; Dpto. de ingeniería de sistemas y automática, Universidad de Málaga.

[15] OpenCV: HighGUI reference manual:
http://www.ai.rug.nl/vakinformatie/pas/content/Highgui/opencvref_highgui.htm
(Última visita Febrero 2011)

[16] Raquel Úbeda, *Aplicación de técnicas de Boosting para detección de matriculas*; Proyecto final de carrera 2001, Universidad Politécnica de Valencia.

[17] Boosting, definición Wikipedia:
http://en.wikipedia.org/wiki/Boosting
(Última visita Febrero 2011)

[18] Viola&Jones, definición Wikipedia:
http://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework
(Última visita Febrero 2011)

[19] Adaboost, definición Wikipedia:
http://en.wikipedia.org/wiki/AdaBoost
(Última visita Febrero 2011)

[20] Receiver Operating Characteristic, definición Wikipedia:
http://en.wikipedia.org/wiki/Receiver_operating_characteristic
(Última visita Febrero 2011)

[21] Juan Carlos González Víctores; *Estudio del Simulador Robótico de Anykode: MARILOU ROBOTICS STUDIO*; Máster en robótica y automatización, Universidad Carlos III de Madrid.

[22] Florian Adolf; *How-to buid a cascade of boosted classifiers based on Haar-like features*; 2003.

[23] OpenCV-performance:
http://manpages.ubuntu.com/manpages/intrepid/man1/opencv-performance.1.html
(Última visita Febrero 2011)

[24] Objectmarker, aplicación y tutorial:
http://www.iem.pw.edu.pl/~domanskj/haarkit.rar
(Última visita marzo 2011)

[25] OpenCV-createsamples:
http://manpages.ubuntu.com/manpages/intrepid/man1/opencv-createsamples.1.html
(Última visita Febrero 2011)

[26] Manual programador OpenCV:
http://carleos.epv.uniovi.es/~nicieza/Documentacion/Manual_Programador_Final.pdf
(Última visita Febrero 2011)

[27] Rapid object detection with a cascade of boosted classifiers based on haar-like features:
http://docs.google.com/View?docID=drw35kw_6gr8r84fs
(Última visita Febrero 2011)

[28] YARP, página oficial:
http://eris.liralab.it/yarp/
(Última visita Abril 2011)

ANEXOS

Anexo I - Tabla 1

Características técnicas cámara IP AXIS 207 MW

Dimensiones	Ancho: 55 mm Profundo: 34 mm Alto: 85 mm
Descripción técnica	Sensor de imagen: CMOS de barrido progresivo 1/4",RGB Micron Objetivo: 4,0 mm/F2.0 iris fijo Sensibilidad lumínica: 1-10,000 Resolución máxima: 640x480 Peso: 190 g.
Características generales	Secuencias de video Motion JPEG de alta calidad Micrófono integrado Servidor Web integrado para una monitorización fácil Interfaz de usuario multilingüe Instalación, configuración y gestión sencillas Seguridad: protección multi-usuario mediante contraseña para restringir el acceso a la cámara.

APÉNDICES

A.I. Función 1

```
void Detection(IplImage* img)
{

  IplImage *gray, *small_img;

  gray = cvCreateImage( cvSize(img->width, img->height), 8, 1);
  small_img = cvCreateImage( cvSize( cvRound (img->width/escala), cvRound (img->height/escala)), 8, 1);

  cvCvtColor( img, gray, CV_BGR2GRAY);
  cvResize( gray, small_img, CV_INTER_LINEAR);
  cvEqualizeHist( small_img, small_img);

  cvClearMemStorage(MemStorage);

  if( classifier )
  {

    CvSeq* bote = cvHaarDetectObjects(small_img, classifier, MemStorage, 1.2, 2, 0, cvSize(50, 65));

        for(int i = 0; i < (bote ? bote->total : 0); i++)
        {

          CvRect* rectangle = (CvRect*)cvGetSeqElem(bote, i);
          CvPoint point1, point2;


          point1.x = cvRound((rectangle->x)*escala);
          point2.x = cvRound((rectangle->x + rectangle->width)*escala);
          point1.y = cvRound((rectangle->y)*escala);
          point2.y = cvRound((rectangle->y + rectangle->height)*escala);

          cvRectangle( img, point1, point2, CV_RGB(255,0,0), 3, 8, 0 );

          CvFont font;
          cvInitFont(&font,CV_FONT_HERSHEY_TRIPLEX,0.6,0.6,0.25);
          char *coords=(char *)malloc(sizeof(char)*20);

          int coord_med_x=(point1.x+point2.x)/2, coord_med_y=(point1.y+point2.y)/2;
          char * ptr_coord_med_x=(char *)malloc(sizeof(char)*20);
          char * ptr_coord_med_y=(char *)malloc(sizeof(char)*20);

          //printf("\ncoord_med_x: %d",coord_med_x);
          //printf("\ncoord_med_y: %d",coord_med_y);

          ptr_coord_med_x=icvst(coord_med_x);
          ptr_coord_med_y=icvst(coord_med_y);

          //printf("\nptr_coord_med_x: %s",ptr_coord_med_x);
          //printf("\nptr_coord_med_y: %s",ptr_coord_med_y);

          sprintf(coords,"(%s,%s)",ptr_coord_med_x,ptr_coord_med_y);


           cvPutText(img,(const char *)coords,cvPoint(point1.x+10,point1.y+rectangle->height
+30),&font,cvScalar(0,215,255));

          free(coords);
        }
    }

  cvShowImage("Result", img);
  cvReleaseImage( &gray);
  cvReleaseImage( &small_img);
}
```

```c
char *icvst(int valor)
{
	int i=0;
	char *pstring=(char *)malloc((i+1)*sizeof(char)),* aux;

	pstring[i]=48+(valor%10);

	do
	{
		i++;
		pstring=(char *)realloc(pstring,(i+1)*sizeof(char));
		valor/=10;
		pstring[i]=48+(valor%10);

	}while(valor/10>0);

	i++;
	pstring[i]='\0';

	aux=(char *)malloc(strlen(pstring)*sizeof(char));

	for(i=0;i<strlen(pstring);i++)
	{
		aux[i]=pstring[strlen(pstring)-1-i];
	}

	aux[i]='\0';

	free(pstring);
	return aux;
}
```

A.II. Programa 1

```
/*****************************BIBLIOTECAS*****************************/
#include "cv.h"
#include "highgui.h"
#include <stdio.h>


/*******************************MAIN*********************************/
int main()
 {

    // Captura la cámara de red con la dirección IP indicada
    CvCapture* capture = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");

    if( !capture )
     {
       fprintf( stderr, "ERROR: capture is NULL \n" );
       getchar();
       return -1;
     }

    // Crea una ventana en la que se mostrarán las imágenes capturadas
    cvNamedWindow( "163.117.201.37", CV_WINDOW_AUTOSIZE );

    // Muestra la imagen en la ventana y repite hasta pulsar la tecla Esc
    while( 1 )
    {
      IplImage* frame = cvQueryFrame( capture );
      if( !frame )
      {
        fprintf( stderr, "ERROR: frame is null...\n" );
        getchar();
        break;
       }

      cvShowImage( "163.117.201.37", frame );

      if( (cvWaitKey(10) & 255) == 27 ) break;
     }

    //Liberar espacio usado de memoria
    cvReleaseCapture( &capture );
    cvDestroyWindow( "163.117.201.37" );

    return 0;
}
```

A.III. Programa 2

```
/*********************************BIBLIOTECAS*********************************/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "cv.h"
#include "highgui.h"

/************DECLARACIONES DE FUNCIONES Y VARIABLES LOCALES************/
static CvMemStorage* MemStorage = 0;
static CvHaarClassifierCascade* classifier = 0;

void Detection(IplImage* image);

const char* cascade_name = "/home/raul/Detector 18/cascade2xml/cascade.xml";
double escala = 1;


/*********************************MAIN*********************************/
int main(int argc, char** argv)
{

  IplImage *frame, *framecopy = 0;

  //Cargar clasificador según xml creado en el entrenamiento.
  classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);

  //Crear memoria de almacenamiento de cálculos
  MemStorage = cvCreateMemStorage(0);

  //Tomar fuente de entrada, en este caso la direccion IP de la camara de red
  CvCapture* inputSource = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");

  //Verificar fuente de entrada y clasificador
  if(!inputSource || !classifier)
    return -1;

  //Crear ventana para mostrar el resultado
  cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);

   if(inputSource)
   {

     for(;;)
     {
        if( !cvGrabFrame(inputSource))
           break;

        //Grabar y almacenar la imagen del video
        frame = cvRetrieveFrame(inputSource);

        if(!frame)
           break;

        //Crear copia modificada de la imagen
        if(!framecopy)
           framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);

        //Si el origen de la imagen está arriba a la izda se copia la imagen en el puntero a copia de
imagen
        //(framecopy), en caso contrario (origen abajo a la izda) la voltea alrededor del eje x
        if(frame->origin == IPL_ORIGIN_TL)
          cvCopy(frame, framecopy, 0);
```

```
        else
          cvFlip(frame, framecopy, 0);

        Detection(framecopy);

        if(cvWaitKey(10) >= 0)
        {
          //Liberar copia de imagen y fuente de entrada

          cvReleaseImage( &framecopy );
          cvReleaseCapture( &inputSource );
         }

    }

    cvWaitKey(0);

  }


  cvDestroyWindow("Result");

  return 0;

}


/*****************************DETECTION*****************************/
void Detection(IplImage* img)
{

  IplImage *gray, *small_img;

  //Crear la primera imagen en escala de grises y la segunga a escala
  gray = cvCreateImage( cvSize(img->width, img->height), 8, 1);
  small_img = cvCreateImage( cvSize( cvRound (img->width/escala), cvRound (img->height/escala)), 8, 1);
  cvCvtColor( img, gray, CV_BGR2GRAY);

  //Cambiar el tamaño de la imagen a escala de grises igual a la imagen a escala y equalizar el
histograma
  cvResize( gray, small_img, CV_INTER_LINEAR);
  cvEqualizeHist( small_img, small_img);

  //Limpiar memoria
  cvClearMemStorage(MemStorage);

  if( classifier )
  {

    //Crear puntero que identifican a los botes
    CvSeq* bote = cvHaarDetectObjects(small_img, classifier, MemStorage, 1.2, 2, 0, cvSize(50, 65));

        for(int i = 0; i < (bote ? bote->total : 0); i++)
        {

          //Obtener las características de un rectángulo
          CvRect* rectangle = (CvRect*)cvGetSeqElem(bote, i);

          CvPoint point1, point2;

          //Obtener las coordenadas del rectángulo
          point1.x = cvRound((rectangle->x)*escala);
          point2.x = cvRound((rectangle->x + rectangle->width)*escala);
          point1.y = cvRound((rectangle->y)*escala);
          point2.y = cvRound((rectangle->y + rectangle->height)*escala);

          //Dibujar el rectángulo en la imagen
          cvRectangle( img, point1, point2, CV_RGB(255,0,0), 3, 8, 0 );

        }
    }
```

```
        //Mostrar la imagen en la ventana
        cvShowImage( "Result", img);

        //Liberar memoria
        cvReleaseImage( &gray);
        cvReleaseImage( &small_img);

}
```

A.IV. Programa 3

```c
/*************************BIBLIOTECAS*************************/
#include <gtk/gtk.h>


/************************ON_BUTTON_CLICKED************************/
void on_button_clicked (GtkButton *button, gpointer
data)
{
    //Entrada
    GtkLabel *label = GTK_LABEL (data);
    const gchar* text = gtk_label_get_text (label);

    //Implementación
    gint i;
    gint t_size = g_utf8_strlen (text, -1);
    gchar* inverted_text = g_new0 (gchar, t_size);
    for (i = 0; i < t_size; i++) {
        inverted_text[i] = text [t_size - i - 1];
    }

    //Salida
    gtk_label_set_text (label, inverted_text);
    g_free (inverted_text);
}


/************************CONSTRUCT_WINDOW************************/
GtkWidget * construct_window (void)
{
    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *button;
    GtkWidget *label;

    //Definición de las variables
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window), "Mi primer ejemplo de GTK+");
    vbox = gtk_vbox_new (TRUE, 0);
    button = gtk_button_new_with_label ("Invierte texto");
    label = gtk_label_new ("Anita lava la tina");

    //Construcción del entorno de la ventana
    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);

    //creacion de la señal encargada de llamar a la funcion al presionar el boton
    g_signal_connect (G_OBJECT (button), "clicked", G_CALLBACK (on_button_clicked), label);

    return window;
}

/*****************************MAIN*****************************/
int main (gint argc, gchar **argv)
{
    //Inicializamos la "toolkit"
    gtk_init (&argc, &argv);

    //Crear y caracterizar ventana
    GtkWidget *window = construct_window ();
    gtk_window_set_default_size (GTK_WINDOW (window), 400, 300);
    gtk_widget_show_all (window);
```

```
    //Crear señal para cerrar la ventana
    g_signal_connect (G_OBJECT (window), "delete-event", gtk_main_quit, NULL);

    //ejecutar el bucle principal hasta dar la orden de salir
    gtk_main ();

    return 0;
}
```

A.V. Programa 4

```
/*************************BIBLIOTECAS*************************/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>
#include "cv.h"
#include "highgui.h"

/*****************DECLARACIONES DE FUNCIONES*****************/
GtkWidget * construct_window (void);
int on_button_clicked (GtkButton *button,gpointer data);


/*****************************MAIN*****************************/
int main (gint argc, gchar **argv)
{
    gtk_init (&argc, &argv);

    GtkWidget *window = construct_window ();
    gtk_window_set_default_size (GTK_WINDOW (window), 600, 600);
    gtk_widget_show_all (window);

    g_signal_connect (G_OBJECT (window), "delete-event",gtk_main_quit, NULL);

    gtk_main ();

    return 0;
}

/************************CONSTRUCT_WINDOW************************/
GtkWidget * construct_window (void)
{
    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *button;

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window),"Detector bote Coca-cola");
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER_ALWAYS);

    vbox = gtk_vbox_new (TRUE, 0);
    button = gtk_button_new_with_label("Camara .37");

    gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);

    g_signal_connect (G_OBJECT (button),"clicked",G_CALLBACK(on_button_clicked),NULL);

    return window;
}
/************************ON_BUTTON_CLICKED************************/
int on_button_clicked (GtkButton *button,gpointer data)
{
    CvCapture* capture = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");

    if( !capture )
     {
      fprintf( stderr, "ERROR: capture is NULL \n" );
      getchar();
      return -1;
     }
```

```
    cvNamedWindow( "163.117.201.37", CV_WINDOW_AUTOSIZE );

    while( 1 )
     {
      IplImage* frame = cvQueryFrame( capture );
      if( !frame )
       {

        fprintf( stderr, "ERROR: frame is null...\n" );
        getchar();
        break;
       }

      cvShowImage( "163.117.201.37", frame );

      if( (cvWaitKey(10) & 255) == 27 ) break;
     }

    cvReleaseCapture( &capture );
    cvDestroyWindow( "163.117.201.37" );

    return 0;
}
```

A.VI. Programa 5

```
/********************************** BIBLIOTECAS ****************************/
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gtk/gtk.h>
#include <gdk/gdk.h>
#include "cv.h"
#include "highgui.h"


/****************************** DECLARACIONES DE FUNCIONES ***************/
GtkWidget * construct_window (void);
void on_button_clicked1 (GtkButton *button,gpointer data);
void on_button_clicked2 (GtkButton *button,gpointer data);
void on_button_clicked3 (GtkButton *button,gpointer data);
void on_button_clicked4 (GtkButton *button,gpointer data);
void on_button_clicked5 (GtkButton *button,gpointer data);
void Detection(IplImage* image);
char *icvst(int valor);

/****************************** VARIABLES GLOBALES ***********************/
static CvMemStorage* MemStorage = 0;
static CvHaarClassifierCascade* classifier = 0;
const char* cascade_name = "/home/raul/Detector 18/cascade2xml/cascade.xml";
const gchar* filename = "/home/raul/Escritorio/gtk/bote/Imagenes/Logo uc3m azul, 160x160.jpg";
double escala = 1;


/*****************************************************************************/
/************************DEFINICIONES DE FUNCIONES ***********************/
/*****************************************************************************/

/************************************ MAIN *******************************/
int main (gint argc, gchar **argv)
{
    gtk_init (&argc, &argv);
    GtkWidget *window = construct_window ();
    gtk_window_set_default_size (GTK_WINDOW (window), 550, 550);
    gtk_widget_show_all (window);
    g_signal_connect (G_OBJECT (window), "delete-event",gtk_main_quit, NULL);
    gtk_main ();
    return 0;
}

/****************************** CONSTRUCT_WINDOW *************************/
GtkWidget * construct_window (void)
{
    GtkWidget *window;
    GtkWidget *vbox;
    GtkWidget *menu;
    GtkWidget *button1, *button2, *button3, *button4, *button5;
    GtkWidget *image;
    GdkColor azul_claro, oro, magenta, lima;

    azul_claro.red=44461;azul_claro.green=55512;azul_claro.blue=59110;
    oro.red=65535;oro.green=55255;oro.blue=0;
    magenta.red=60909;magenta.green=21845;magenta.blue=31097;
    lima.red=0;lima.green=65535;lima.blue=0;
```

```
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (window),"DETECTOR bote Coca-cola");
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER_ALWAYS);
    gtk_widget_modify_bg(window, GTK_STATE_NORMAL,&azul_claro);

    gtk_container_set_border_width (GTK_CONTAINER (window), 70);

    vbox = gtk_vbox_new (FALSE, 0);
    image = gtk_image_new_from_file(filename);
    menu = gtk_label_new("MENU USUARIO");
    button1 = gtk_button_new_with_label("Camara _3_7");
    gtk_button_set_use_underline(GTK_BUTTON(button1),TRUE);

    gtk_widget_modify_bg(button1, GTK_STATE_NORMAL,&oro);
    button2 = gtk_button_new_with_label("Camara _3_8");
    gtk_button_set_use_underline(GTK_BUTTON(button2),TRUE);
    gtk_widget_modify_bg(button2, GTK_STATE_NORMAL,&oro);
    button3 = gtk_button_new_with_label("Camara _3_9");
    gtk_button_set_use_underline(GTK_BUTTON(button3),TRUE);
    gtk_widget_modify_bg(button3, GTK_STATE_NORMAL,&oro);
    button4 = gtk_button_new_with_label("Web_c_a_m");
    gtk_button_set_use_underline(GTK_BUTTON(button4),TRUE);
    gtk_widget_modify_bg(button4, GTK_STATE_NORMAL,&magenta);
    button5 = gtk_button_new_with_label("Salir");
    gtk_widget_modify_bg(button5, GTK_STATE_NORMAL,&lima);

    gtk_box_pack_start (GTK_BOX (vbox), image, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox), menu, FALSE, FALSE, 34);
    gtk_box_pack_start (GTK_BOX (vbox), button1, FALSE, FALSE, 12);
    gtk_box_pack_start (GTK_BOX (vbox), button2, FALSE, FALSE, 12);
    gtk_box_pack_start (GTK_BOX (vbox), button3, TRUE, FALSE, 12);
    gtk_box_pack_start (GTK_BOX (vbox), button4, TRUE, FALSE, 22);
    gtk_box_pack_start (GTK_BOX (vbox), button5, TRUE, FALSE, 51);

    gtk_container_add (GTK_CONTAINER (window), vbox);

    g_signal_connect (G_OBJECT (button1),"clicked",G_CALLBACK(on_button_clicked1),NULL);
    g_signal_connect (G_OBJECT (button2),"clicked",G_CALLBACK(on_button_clicked2),NULL);
    g_signal_connect (G_OBJECT (button3),"clicked",G_CALLBACK(on_button_clicked3),NULL);
    g_signal_connect (G_OBJECT (button4),"clicked",G_CALLBACK(on_button_clicked4),NULL);
    g_signal_connect (G_OBJECT (button5),"clicked",G_CALLBACK(on_button_clicked5),NULL);

    return window;
}

/********************************* ON_BUTTON_CLICKED1 ***********************/
void on_button_clicked1 (GtkButton *button,gpointer data)
{
  IplImage *frame, *framecopy = 0;

  classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);
  MemStorage = cvCreateMemStorage(0);

  CvCapture* inputSource = cvCreateFileCapture("http://root:root@163.117.201.37/mjpg/video.mjpg");

  if(!inputSource || !classifier)
    exit(1);

  cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);

  if(inputSource)
  {
    for(;;)
    {
       if( !cvGrabFrame(inputSource))
          break;

       frame = cvRetrieveFrame(inputSource);

       if(!frame)
          break;
```

```
       if(!framecopy)
          framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);

       if(frame->origin == IPL_ORIGIN_TL)
         cvCopy(frame, framecopy, 0);
       else
         cvFlip(frame, framecopy, 0);

       Detection(framecopy);

       if(cvWaitKey(10) >= 0)
       {

         cvReleaseImage( &framecopy );
         cvReleaseCapture( &inputSource );
        }
    }
    cvWaitKey(0);
  }
  cvDestroyWindow("Result");
}

/******************************** ON_BUTTON_CLICKED2 **********************/
void on_button_clicked2 (GtkButton *button,gpointer data)
{
  IplImage *frame, *framecopy = 0;

  classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);
  MemStorage = cvCreateMemStorage(0);

  CvCapture* inputSource = cvCreateFileCapture("http://root:root@163.117.201.38/mjpg/video.mjpg");

  if(!inputSource || !classifier)
    exit(1);

  cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);

  if(inputSource)
  {
    for(;;)
    {
       if( !cvGrabFrame(inputSource))
          break;

       frame = cvRetrieveFrame(inputSource);

       if(!frame)
          break;

       if(!framecopy)
          framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);

       if(frame->origin == IPL_ORIGIN_TL)
         cvCopy(frame, framecopy, 0);
       else
         cvFlip(frame, framecopy, 0);

       Detection(framecopy);

       if(cvWaitKey(10) >= 0)
       {
         cvReleaseImage( &framecopy );
         cvReleaseCapture( &inputSource );
        }
    }
    cvWaitKey(0);
  }
  cvDestroyWindow("Result");
}
```

```
/******************************** ON_BUTTON_CLICKED3 **********************/
void on_button_clicked3 (GtkButton *button,gpointer data)
{
  IplImage *frame, *framecopy = 0;

  classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);
  MemStorage = cvCreateMemStorage(0);

  CvCapture* inputSource = cvCreateFileCapture("http://root:root@163.117.201.39/mjpg/video.mjpg");

  if(!inputSource || !classifier)
    exit(1);

  cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);

   if(inputSource)
   {
     for(;;)
     {
        if( !cvGrabFrame(inputSource))
           break;

        frame = cvRetrieveFrame(inputSource);

        if(!frame)
           break;

        if(!framecopy)
           framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);

        if(frame->origin == IPL_ORIGIN_TL)
          cvCopy(frame, framecopy, 0);
        else
          cvFlip(frame, framecopy, 0);

        Detection(framecopy);

        if(cvWaitKey(10) >= 0)
        {
          cvReleaseImage( &framecopy );
          cvReleaseCapture( &inputSource );
         }
     }
     cvWaitKey(0);
   }
   cvDestroyWindow("Result");
}

/******************************** ON_BUTTON_CLICKED4 **********************/
void on_button_clicked4 (GtkButton *button,gpointer data)
{
  IplImage *frame, *framecopy = 0;

  classifier = (CvHaarClassifierCascade*) cvLoad(cascade_name, 0, 0, 0);
  MemStorage = cvCreateMemStorage(0);

  CvCapture* inputSource = cvCaptureFromCAM(0);

  if(!inputSource || !classifier)
    exit(1);

  cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);

  if(inputSource)
  {
    for(;;)
    {
       if( !cvGrabFrame(inputSource))
          break;

       frame = cvRetrieveFrame(inputSource);
```

```
       if(!frame)
          break;

       if(!framecopy)
          framecopy = cvCreateImage( cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame-
>nChannels);

       if(frame->origin == IPL_ORIGIN_TL)
         cvCopy(frame, framecopy, 0);
       else
         cvFlip(frame, framecopy, 0);

       Detection(framecopy);


       if(cvWaitKey(10) >= 0)
       {
         cvReleaseImage( &framecopy );
         cvReleaseCapture( &inputSource );
        }
    }
    cvWaitKey(0);
  }
  cvDestroyWindow("Result");
}

/******************************** ON_BUTTON_CLICKED5 **********************/
void on_button_clicked5 (GtkButton *button,gpointer data)
{

        gtk_main_quit();

}

/******************************** DETECTION_IMAGE **************************/
void Detection(IplImage* img)
{

  IplImage *gray, *small_img;

  gray = cvCreateImage( cvSize(img->width, img->height), 8, 1);
  small_img = cvCreateImage( cvSize( cvRound (img->width/escala), cvRound (img->height/escala)), 8, 1);

  cvCvtColor( img, gray, CV_BGR2GRAY);
  cvResize( gray, small_img, CV_INTER_LINEAR);
  cvEqualizeHist( small_img, small_img);

  cvClearMemStorage(MemStorage);

  if( classifier )
  {

    CvSeq* bote = cvHaarDetectObjects(small_img, classifier, MemStorage, 1.2, 2, 0, cvSize(50, 65));

        for(int i = 0; i < (bote ? bote->total : 0); i++)
        {

          CvRect* rectangle = (CvRect*)cvGetSeqElem(bote, i);
          CvPoint point1, point2;


          point1.x = cvRound((rectangle->x)*escala);
          point2.x = cvRound((rectangle->x + rectangle->width)*escala);
          point1.y = cvRound((rectangle->y)*escala);
          point2.y = cvRound((rectangle->y + rectangle->height)*escala);

          cvRectangle( img, point1, point2, CV_RGB(255,0,0), 3, 8, 0 );

          CvFont font;
          cvInitFont(&font,CV_FONT_HERSHEY_TRIPLEX,0.6,0.6,0.25);
          char *coords=(char *)malloc(sizeof(char)*20);
```

```
            int coord_med_x=(point1.x+point2.x)/2, coord_med_y=(point1.y+point2.y)/2;
            char * ptr_coord_med_x=(char *)malloc(sizeof(char)*20);
            char * ptr_coord_med_y=(char *)malloc(sizeof(char)*20);

            //printf("\ncoord_med_x: %d",coord_med_x);
            //printf("\ncoord_med_y: %d",coord_med_y);

            ptr_coord_med_x=icvst(coord_med_x);
            ptr_coord_med_y=icvst(coord_med_y);

            //printf("\nptr_coord_med_x: %s",ptr_coord_med_x);
            //printf("\nptr_coord_med_y: %s",ptr_coord_med_y);

            sprintf(coords,"(%s,%s)",ptr_coord_med_x,ptr_coord_med_y);


           cvPutText(img,(const char *)coords,cvPoint(point1.x+10,point1.y+rectangle->height
+30),&font,cvScalar(0,215,255));

          free(coords);
        }
    }

  cvShowImage("Result", img);
  cvReleaseImage( &gray);
  cvReleaseImage( &small_img);
}

/******************************** INT CONVERT STRING *************************/

char *icvst(int valor)
{
        int i=0;
        char *pstring=(char *)malloc((i+1)*sizeof(char)),* aux;

        pstring[i]=48+(valor%10);

        do
        {
                i++;
                pstring=(char *)realloc(pstring,(i+1)*sizeof(char));
                valor/=10;
                pstring[i]=48+(valor%10);

        }while(valor/10>0);

        i++;
        pstring[i]='\0';

        aux=(char *)malloc(strlen(pstring)*sizeof(char));

        for(i=0;i<strlen(pstring);i++)
        {
                aux[i]=pstring[strlen(pstring)-1-i];
        }

        aux[i]='\0';

        free(pstring);
        return aux;
}
```